

tiny C
owners
manual

tiny
C

tiny

overmagnets

tiny C

machine/os *POLY-88 / MONITOR 4.0*
media/format *BYTE STANDARD CASSETTE*
version *80-01-01*

note: the enclosed guide contains instructions for installing tiny-c; it is not a substitute for the documentation available in the tiny-c owner's manual.

tiny
C

tiny c associates
post office box 269
holmdel, new jersey 07733

1 NOV
1979

ALWAYS
ZIP



Rob Bohu, Jr.
1006 Wayne
North Manchester, IN
46962

D

tiny c associates is happy to announce that we are now carrying the CP/M-compatible **BDS C compiler**. This compiler handles the full C language with the exception of the floating point data type.

The compiler package includes a linking loader which enables you to combine compiled C programs with your own routines. You also receive a 40-page user's manual for a total package price of \$100.00.

For an additional \$10.00, we will also include a copy of the Kernighan and Ritchie paperback, **The C Programming Language**.

We welcome telephone orders billed to VISA or MASTERCHARGE.

tiny c associates, p.o. box 269, holmdel, new jersey 07733 (201) 671-2296



tiny-c price list & order blank

	PRICE	QTY.	TOTAL
tiny-c two, including Owner's Manual, CP/M 8" diskette, tiny-shell, all source code.	\$250.00	_____	_____
tiny-c two, Owner's Manual Only	\$50.00	_____	_____
tiny-c two, diskette only with source	\$200.00	_____	_____
tiny-c one, including Owner's Manual with training, reference, theory, and choice of diskette. All diskettes have the tiny-c inter- preter, and the Program Preparation System; all except North Star have all source codes.	\$100.00	_____	_____

20% DISCOUNT
to tiny c one owners.

Indicate Choices:

	QTY.
8" CP/M	_____
5" North Star CP/M	_____
5" Micropolis CP/M double density	_____
5" Micropolis CP/M quad density	_____
PDP-11/RT-11	_____
Apple II DOS 3.2	_____
H8/H89 HDOS	_____
CDOS with z-80 code	_____
6800/Flex 2.0	_____
North Star 5"	_____
8080 source code on PDP/11 diskette	_____

tiny-c one, including Owner's Manual
and choice of cassette. All cassettes have the
tiny-c interpreter and the Program Prepara-
tion System.
Indicate Choice:

	QTY.
TRS - 80 Level II	_____
KIM 6502	_____
SYM 6502	_____
Tarbell 8080	_____
SOL 20 CUTS 8080	_____

tiny-c one, Owner's Manual Only	\$50.00	_____	_____
tiny-c one, diskette only (indicate choice above)	\$50.00	_____	_____
tiny-c one, cassette only (indicate choice above)	\$50.00	_____	_____

More on other side

Total on this side _____

tiny-c one, listing of interpreter
source code. Indicate choice:

\$15.00 _____

QTY.

TRS-80/Z-80 _____

CDOS/Z-80 _____

6502 _____

6800/Flex 2.0 _____

PDP-11 _____

8080-01-01 (Tarbell, Sol 20, North Star) _____

8080-01-02 (CP/M, HDOS) _____

tiny-c in C _____

C COMPILERS

BDS C compiler for 8080 CP/M

\$135.00 _____

- includes linker, C library, & librarian

QTY.

8" CP/M _____

5" North Star CP/M double density _____

5" Micropolis CP/M double density _____

5" Micropolis CP/M quad density _____

C/80 C compiler for Heath H-8 & H-89, Zenith Z-89

\$39.95 _____

- C library & runtime profile facility included
- generates Heath/Zenith assembly code
- 5" HDOS diskette

BOOKS

The C Programming Language

\$13.95 _____

- by Brian Kernigham & Dennis Ritchie

Purchased with BDS C compiler

\$10.00 _____

C Notes

\$15.00 _____

- by Carroll Zahn

Dr. Dobb's C Issue, May '80

\$3.00 _____

Total for this side _____

Total from other side _____

Subtotal _____

New Jersey residents add 5% sales tax _____

Air Mail to Europe, \$20 Owner's Manual _____

Grand Total _____

Send to: tiny c associates
Post Office Box 269
Holmdel, New Jersey 07733
(201) 671-2296

_____ check enclosed

_____ Master Charge No. _____

Expiration Date _____

_____ Bank Americard/VISA No. _____

Expiration date _____

Name _____

Street _____

City _____ State _____ Zip Code _____

November 1, 1980

DESCRIPTION =====	PRICE =====	QUANTITY =====	TOTAL =====
tiny-c Owner's Manual *training, reference, theory	\$40 ea.	-----	-----
Poly-88 Byte Standard Cassette *graphics & printer support	\$25 ea.	-----	-----
Tarbell 8080 Cassette *standard Tarbell console interface	\$20 ea.	-----	-----
PDP-11 Diskette, Version 11-01-02A *RT-11 load-and-go; includes source	\$25 ea.	-----	-----
CP/M Diskettes with 8080 Source			
*8" diskette	\$35 ea.	-----	-----
*5" Micropolis CP/M double density	\$35 ea.	-----	-----
*5" Micropolis CP/M quad density	\$35 ea.	-----	-----
TRS-80 Level II SYSTEM Format Cassette *EDTASM-compatible file I/O *line printer and graphics support *16K required	\$30 ea.	-----	-----
8080 Source on PDP-11 Diskette *includes macros to assemble w/ MACRO-11	\$35 ea.	-----	-----
The C Programming Language *by Kernighan and Ritchie *official C language reference	\$10.95 ea.	-----	-----
TOTALS from other side			-----
SUBTOTAL			-----
New Jersey residents add 5% sales tax			-----
TOTAL			-----

Send to: tiny c associates
 Post office box 269
 Holmdel, New Jersey 07733
 (201) 671-2296

_____check enclosed
 _____Master Charge No. _____Expiration date _____
 _____BankAmericard/Visa _____Expiration date _____

NAME _____

STREET _____

CITY, STATE, ZIP _____

COMPUTER SYSTEM _____

ALSO AVAILABLE

DESCRIPTION

PRICE

QUANTITY

TOTAL

*SOL 20
installation by: METRON COMPUTERWARE INC.
Post Office Box 865
New York, New York 10025
(212) 666-2400

Helios 8080 Diskette
and

CUTS 8080 Cassette

*SOL 20 load-and-go
*Program Preparation System
arranged for easy editing

\$35 ea.

\$30 ea.

*NORTH STAR

installation also by: METRON COMPUTERWARE INC.

North Star 5" Diskette

*single density
*loaded at 2A00

\$35 ea.

TOTAL (Please record on Side 1)

COMING SOON

Digital Group

Data General

tiny c

**A new structured
programming language**

**tiny
c**


```

/* Guess a number between 1 and 100.
/* T. A. Gibson, 11/29/76
guessnum [
    int guess, number
    number = random (1,100)
    pl "Guess a number between 1 and 100"
    pl "Type in your guess now"
    while (guess != number) [
        guess = gn
        if (guess == number) pl "right!!"
        else if (guess > number) pl "too high"
        else if (guess < number) pl "too low"
    ]
]
/* End of game loop
/* End of program

/* random — generates a random number between
/* little and big
int seed, last /* Globals used by random
random int little, big [
    int range
    if (seed == 0) seed = last = 99
    range = big - little + 1
    last = last * seed
    if (last < 0) last = -last
    return little + (last/8) % range
]

```


This is a tiny-c program. How does it work? The first two lines are comments. A comment begins with `/*` and goes to the end of the line.

The next 12 lines define a function called "guessnum". Its definition is enclosed in the brackets `[...]`. Two integer variables, "guess" and "number" are created by the "int" statement. Then a random number between 1 and 100 is put into number. That's the number the user must guess.

`pl` means print on a new line. It prints the message between the quotes. There are two `pl`'s, so two lines are printed for the user to read. On the console is displayed:

```
Guess a number between 1 and 100
Type in your guess now
```

Next the program enters a while loop. At this time guess is zero, and number is something between 1 and 100. The while condition (`guess != number`) is evaluated. `!=` means "not equal to". Indeed, zero is not equal to something between 1 and 100, so the while condition is true. Therefore, the statements "guess = gn" through "if (`guess < number`) ... " are done. In fact, they are done repeatedly until guess and number become equal.

`gn` means get a number. It reads a number typed by the user. The statement "guess = gn" reads a number typed by the user and puts it into guess. The next three statements test if the user's guess is equal to (`==`) number, greater than (`>`) number, or less than (`<`) number. One of these is true, and an appropriate message is printed by a `pl`. Then the first `]` is reached; it ends the set of statements executed during the time the while condition is true. It also causes the while statement to be repeated, because whiles are executed again and again until their condition becomes false.

So "guess != number" is evaluated again. This time it may be true, or it may be false, depending on what the user's guess was. If it is true, the user has not yet guessed the number, and the game continues another round. If it is false, then the user has guessed the number, and the bracketed statements are skipped. This brings the program to the second `]`. This one is the end of guessnum. So the program stops. The game is over.

What are the rest of the lines? Well, they define the function named "random" used by guessnum. A walkthrough of random is contained in the tiny-c Owner's Manual.



LANGUAGE

Data Types	integer, character, pointer
Arrays	one-dimensional
Statements	if-else, while, break, return, compound, assignment
Function Calls	arguments, recursive, returned value
Variable Name Length	unlimited
Multiple Assignment	yes
Variable Scope	local and global
Program Format	free form
Language Type	structured, interpreted
Pointer Expressions	yes

INTERPRETER

Approximate Size	5k bytes
I/O Interfaces	getchar, putchar, fileopen, fileread, fwrite, fclose, initial program load
Standard Machine Call Library	14 functions, 1.7k bytes
Commented Source Code	yes
Private Machine Calls Possible	yes

PROGRAM PREPARATION SYSTEM

Approximate Size	4k bytes
Source Language	tiny-c
Capabilities	write, edit, run, debug, store, read, link
Program Development Tools	25 standard library functions
Error Diagnostics	type and location
Program Levels	system and application

OWNER'S MANUAL

Size	368 pages
Complete Programs	10, including PPS
Source Code	8080 and PDP-11
Detailed Installation Guides	yes
Theory of Operation Section	yes
Index	yes

tiny c associates
post office box 269
holmdel, new jersey 07733
(201) 671-2296

ABOUT tiny-c one and two

(General information about both versions)

What is tiny-c?

Tiny-c is a structured programming language. It is an alternative to BASIC. Using tiny-c instead of BASIC means you will get your software projects done more quickly and more pleasantly.

Why structured programming?

Programming is an error - prone human task. Structured programming however, less error prone. There just aren't as many ways that nasty, hard-to-find bugs can creep into a structured program. BASIC does not offer structured programming. tiny-c does.

The important elements of structured programming are:

Long variable names -- Naming a variable "klingsleft" is much more suggestive as to its role than naming it "k9".

Functions -- Programs are constructed as modules, each with a well defined role in the whole software project. This is better than pages and pages of statements.

Local variables -- A local variable is protected. Only the function that owns it can change it or use it. this is better than letting any line of code anywhere in a large program change any variable, often by accident.

tiny-c has all these features, and more to make it a pleasant, productive tool in your software tool-kit.

Who is tiny-c for?

- The serious programmer
- The writer of large programs
- The assembler language programmer
- The student looking beyond BASIC
- The educator desiring to teach structured programming
- The user of BASIC looking for something better
- The hobbyist who has heard of structured programming and wants to give it a try



Where did tiny-c come from?

tiny-c is a derivative of a big programming language called C, which was invented at Bell Labs. C was used to program the well known UNIX™ operating system. C is widely used in universities and industry.

What versions of tiny-c are available?

tiny-c comes in two versions:

tiny-c ONE is a training version. It is an interpreter, and runs about as fast as most BASIC's. It comes complete with a Program Preparation System, which is used to prepare programs, run them, change them, save and recall them from cassette or diskette. tiny-c one is available on a wide variety of cassettes and diskettes.

tiny-c TWO is a compiler version. It compiles tiny-c programs into an intermediate code which is interpreted by a Run Time System. It runs 10 times faster than tiny-c one, hence about as fast as the faster compiled BASIC's. tiny-c two has lots of extra features not included with tiny-c one.

Anything else?

- tiny-c products come with all source code, so you know what you're getting. The software student can study these to learn how real compilers and interpreters are constructed. The advanced programmer can add their own features.
- tiny-c documentation is the best in the industry. This is not an idle boast. Ask any of our customers.
- tiny-c products are portable. The real software guru can, with reasonable effort, get tiny-c one or two running on a new processor, or new operating system.



Tiny-C Two — The Compiler

Tiny-c two is ten times faster than tiny-c one. It has many extra features, including long (32 bit) integers, lots of new operators, and redirectable and direct access input/output. This version of tiny-c is viable for professional work, either systems programming or business applications.

It comes with a UNIX™ style command interpreter called the "tiny-shell"™. With the tiny-shell, every compiled tiny-c program becomes a new shell command. Tiny-shell commands can have arguments, and dash (-) options, just as real UNIX shell commands do. The < and > input/output redirection operators are supported.

There are over fifty standard library functions, and this set is readily extended. The input/output functions are UNIX style, including fopen, fprintf, etc. Both ascii and raw (binary) input/output are supported.

And the entire package is portable. Bringing it up on a new processor or new operating system should take a few days or a few weeks at the most. And as usual with tiny-c products, all the source code is included.

LANGUAGE FEATURES

- All the features of tiny-c one
- Long (32 bit) integers as well as regular (16 bit) integers
- Additional operators: not, complement, address of, postfix and prefix increment and decrement, left and right shift, and, or, exclusive or
- UNIX style i/o: redirectable by the tiny-shell or by program, ascii and raw (binary), formatted print and scan, direct access (lseek)
- Program chaining for very large applications
- Dynamic storage allocation (calloc, cfree)
- Improved machine language interface

PHYSICAL FEATURES

- 32K recommended. This is enough to compile the compiler
- The compiler is written in tiny-c; all source code is included
- Emits a very compact, stack oriented intermediate code
- Interpreter for the intermediate code uses about 2K bytes
- Standard assembly language portion of the library uses about another 2K bytes. (The tiny-c coded portion of the library is loaded as needed.)
- PORTABLE -- readily transported to other processors or operating systems. The bootstrap procedure is well documented, and tests are provided.
- Speed: 500 to 1000 statements per second on typical 2 MHz to 4 MHz 8 bit processors

HUMAN FEATURES

- Thorough documentation: over 200 pages. This included a tutorial walk-through, a reference chapter, a reference with examples on the tiny-shell, lots of sample programs (and they are useful ones), internals describing how the compiler and linker interface to the tiny-shell, and all the details on how to install this system on any computer.
- The tiny-shell supports multiple commands per line, input/output redirection, and has thorough error control. Most commands have UNIX style dash (-) options.
- Structured programming means you get more code working in a session, and the code is more readily understood by people.



Quickie Index

Statements	- 2-17
Standard Library	- 2-21
PPS Commands	- 3-3
Program error codes	- 3-10
File system error codes	- 6 P9 (12)

To load a file written by ASM G03, change 33E7H to 00. That allows comment files to not cause an error, and change 33F4 to C3 J33ED which changes the Auto execute record to a plain EOF.

L33E7U0U

L33F4UC3 J33EDU

MC 10 is an 0FFH for debugging purposes.

2022	PRBEGIN	NOP/NOP/RET	begin program	} see 6.5.4.3
2025	STBEGIN	NOP/NOP/RET	begin a statement	
2028	PRDONE	NOP/NOP/RET	end program	

Published by tiny-c associates, Post Office Box 288,
Holmdel, New Jersey 07733. Prepared on an LSI-11 processor
running a modified version of the Life Support System
Group, and set on a Digidic 1000.

First Printing: June 1978

000553

tiny-c OWNER'S MANUAL

A Home Computing Software System

Thomas A. Gibson

and

Scott B. Guthery

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of the publisher. With the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES

The authors have taken due care in preparing this book and the program in it, including research, development, and testing to ascertain their effectiveness. The authors and tiny-c associates make no expressed or implied warranty of any kind with regard to these programs or the accompanying documentation in this book. In no event shall tiny-c associates be liable for incidental or consequential damages in connection with or arising out of the furnishing, performance or use of any of these programs.

© tiny-c associates 1978

Published by tiny-c associates, Post Office Box 269, Holmdel, New Jersey 07733. Prepared on an LSI-11 processor running a modified version FORMAT by the Life Support System Group, and set on a Diablo 1620.

First Printing: June 1978

000559

© tiny-c associates 1978

All rights reserved. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise without the prior written permission of the publishers, with the exception that the program listings may be entered, stored, and executed in a computer system, but they may not be reproduced for publication.

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES

The authors have taken due care in preparing this book and the programs in it, including research, development, and testing to ascertain their effectiveness. The authors and tiny-c associates make no expressed or implied warranty of any kind with regard to these programs nor the supplementary documentation in this book. In no event shall the authors or tiny-c associates be liable for incidental or consequential damages in connection with or arising out of the furnishing, performance or use of any of these programs.

tiny-c OWNER'S MANUAL
Version 1.00

TABLE OF CONTENTS

ACKNOWLEDGMENTS	xi
PREFACE	xiii
FOREWORD by Brian W. Kernighan	xv
 I. INTRODUCTION	 1-1
1.1 A Tiny-c Program Walk-Through	1-1
1.2 Structured Programming -- What Tiny-c Is All About	1-7
1.2.1 Compound Statements	1-8
1.2.2 Nesting Compound Statements	1-10
1.2.3 Readable Program Flow	1-12
1.2.4 Indenting and the Placement of Brackets in Compound Statements	1-13
1.2.5 Functions	1-14
1.2.6 Local and Global Variables	1-16
1.2.7 Summary -- And Where We Go from Here	1-18
 II. THE LANGUAGE	 2-1
2.1 Comments	2-1
2.2 Names	2-1
2.3 Data and Variables	2-2
2.3.1 Arrays and Array Elements	2-2
2.3.2 Locals, Globals, and Arguments	2-3
2.4 Expressions	2-5
2.4.1 Primaries	2-5
2.4.2 Operators	2-6

tiny-c OWNER'S MANUAL
Version 1.00

2.5	Functions	2-10
2.6	Pointers	2-14
2.7	Statements	2-17
2.8	Notes on Using the Statements	2-18
2.9	Libraries and Libraries and Libraries	2-19
2.9.1	Standard Library	2-21
2.9.2	Notes on Using the Standard Library Functions	2-26
2.10	Machine Language Interface	2-29
2.11	Computer Arithmetic	2-34
III.	THE PROGRAM PREPARATION SYSTEM (PPS)	3-1
3.1	Fundamentals of PPS	3-1
3.2	Entering Text Lines	3-3
3.3	The PPS Commands	3-3
3.4	Notes on Using PPS	3-6
3.4.1	Bumping the Top and Bottom	3-6
3.4.2	Deleting	3-7
3.4.3	Line Numbers	3-7
3.4.4	Using Locate and Change	3-7
3.5	Errors	3-9
3.6	Sample Session with PPS	3-11
IV.	TINY-C PROGRAM EXAMPLES	4-1
4.1	Optional Library Functions	4-1
4.1.1	Tiny-c Code for the Optional Library	4-2
4.1.2	Comments on Style	4-4

tiny-c OWNER'S MANUAL
Version 1.00

4.2	Piranha Fish -- An Original Game	4-5
4.2.1	Facts	4-6
4.2.2	Piranha Fish Code	4-9
4.2.3	Comments on Style	4-17
4.3	The Standard Library and PPS	4-17
4.3.1	Program Preparation System Code	4-18
4.3.2	Comments on Style	4-30
4.3.3	The Use of MC 11	4-30
4.4	Morse Code Generator	4-31
4.4.1	Comments on Style	4-34
4.5	A Tape-to-Printer Copy Utility	4-34
4.6	TV Graphics Functions	4-35
4.6.1	Meteor Shower	4-36

V. INTERNAL OPERATION OF THE TINY-C INTERPRETER . . . 5-1

5.1	Data Objects	5-2
5.1.1	Name	5-2
5.1.2	Class	5-3
5.1.3	Size	5-3
5.1.4	Length	5-3
5.1.5	Address	5-4
5.2	The Variable Table	5-4
5.3	Layers in the Variable Table	5-6
5.4	The Function Table	5-8
5.5	Symbol Table Tools	5-8
5.5.1	NEWFUN	5-9
5.5.2	FUNDONE	5-9
5.5.3	NEWVAR	5-9
5.5.4	ADDRVAL	5-10
5.5.5	CANON	5-10

tiny-c OWNER'S MANUAL
Version 1.00

5.6	The Tiny-c Stack	5-10
5.7	Stack Tools	5-12
5.7.1	PUSH	5-12
5.7.2	PUSHK (8080)	5-13
5.7.3	PZERO, PONE (8080)	5-13
5.7.4	POP (8080)	5-13
5.7.5	POP (PDP-11)	5-13
5.7.6	TOPTOI	5-13
5.7.7	POPTWO (8080)	5-14
5.7.8	TOPDIF (8080)	5-14
5.7.9	EQ	5-14
5.8	Scanning Tools	5-14
5.8.1	LIT	5-15
5.8.2	SKIP	5-15
5.8.3	SYMNAME	5-15
5.8.4	CONST	5-16
5.8.5	REM	5-16
5.8.6	BLANKS	5-17
5.9	The Statement Analyzer	5-17
5.9.1	ST	5-21
5.9.2	VALLOC	5-21
5.9.3	ASGN	5-21
5.9.4	RELN	5-22
5.9.5	EXPR	5-22
5.9.6	TERM	5-22
5.9.7	FACTOR	5-22
5.9.8	ENTER	5-23
5.9.9	SETARG	5-23
5.9.10	SKIPST	5-23
5.9.11	LINK	5-24
5.10	Standard Machine Calls	5-24
5.10.1	MC 1 ... PUTCHAR	5-24
5.10.2	MC 2 ... GETCHAR	5-25
5.10.3	MC 3 ... FOPEN	5-25
5.10.4	MC 4 ... FREAD	5-25
5.10.5	MC 5 ... FWRITE	5-25
5.10.6	MC 6 ... FCLOSE	5-25

tiny-c OWNER'S MANUAL
Version 1.00

5.10.7	MC 7	... MOVEBL	5-26
5.10.8	MC 8	... COUNTCH	5-26
5.10.9	MC 9	... SCANN	5-26
5.10.10	MC 10	... INTERRUPT	5-26
5.10.11	MC 11	... ENTER	5-26
5.10.12	MC 12	... CHRDY (8080)	5-28
5.10.13	MC 13	... PFT	5-28
5.10.14	MC 14	... PN	5-28
5.11 Other Tools			5-28
5.11.1	ALPHA		5-28
5.11.2	ALPHANUM		5-29
5.11.3	ATOI		5-29
5.11.4	BZAP (8080)		5-29
5.11.5	CEQ		5-29
5.11.6	CEQN		5-30
5.11.7	CTOI (PDP-11)		5-30
5.11.8	ESET		5-30
5.11.9	MOVN (PDP-11)		5-30
5.11.10	NUM		5-31
5.11.11	RETURN (PDP-11)		5-31
5.11.12	SAVE (PDP-11)		5-31
5.11.13	TCTOI (PDP-11)		5-31
5.11.14	ZERO (8080)		5-32
5.12 16-Bit Arithmetic Tools (8080)			5-32
5.12.1	DNEG		5-32
5.12.2	DENEG		5-32
5.12.3	HLNEG		5-32
5.12.4	BCRS		5-32
5.12.5	DELS		5-33
5.12.6	RDEL		5-33
5.12.7	HLLS		5-33
5.12.8	DADD		5-33
5.12.9	DSUB		5-33
5.12.10	DMPY		5-33
5.12.11	DDIV		5-34
5.12.12	DREM		5-34
5.12.13	DCMP		5-34

VI. INSTALLING TINY-C ON YOUR 8080 SYSTEM	6-1
6.1 STEP 1 -- The Interface Routines	6-2
6.1.1 Specification of the 8080 Terminal Interface	6-3
6.1.2 Specification of the 8080 File Interface	6-3
6.1.3 File System Implementation Options . . .	6-5
6.1.4 Example of a Tiny-c Cassette File System Interface	6-6
6.2 STEP 2 -- Load and Relocate Tiny-c	6-8
6.2.1 Reading the Tiny-c File	6-9
6.2.2 Address Adjustment	6-9
6.3 STEP 3 -- Load the Interface Routines . . .	6-17
6.4 STEP 4 -- Link Tiny-c to the Interface Routines	6-17
6.5 STEP 5 -- Modify the Other Installation Vector Elements	6-17
6.5.1 Upper Case Option	6-17
6.5.2 Allocation of Memory	6-18
6.5.3 8080 Stack	6-20
6.5.4 Optional Entries in the Installation Vector	6-21
6.5.4.1 User Machine Call Jump Address	6-21
6.5.4.2 Choice of Escape	6-22
6.5.4.3 Statement Control Opportunities	6-22
6.6 STEP 6 (and 6') -- Write to Disk Or Tape . .	6-23
6.7 STEP 7 -- Test the Results So Far	6-23
6.7.1 Null Program Test	6-24
6.7.2 Simple Program Test, Hand Loaded . . .	6-24
6.7.3 Simple Program Test, File Loaded . . .	6-26

tiny-c OWNER'S MANUAL
Version 1.00

6.8	STEP 8 -- Prepare PPS for Loading	6-26
6.8.1	Possibly Required PPS Changes	6-28
6.8.2	Optional PPS Changes	6-28
6.9	Write the "Load-And-Go" Tiny-c to Mass Storage	6-29
6.10	If Things Go Wrong	6-29
6.11	Key Points of a Tiny-c Installation	6-31
6.12	Modifying the Program Preparation System	6-32
6.12.1	The System Loader	6-33
6.12.2	Debugging a New PPS	6-34
VII.	INSTALLING TINY-C ON YOUR PDP-11 SYSTEM	7-1
7.1	Logical Division of Tiny-c/11	7-1
7.2	The Installation Vector	7-2
7.2.1	Input/Output Routine Calls	7-2
7.2.2	User Option Routines	7-5
7.2.3	Working Areas	7-6
7.3	The Sizing Constants	7-6
7.4	Subroutine Call and Processor Stack Regimen	7-7
7.4.1	Local Stack Structure	7-9
7.4.2	Subroutine Calling	7-9
7.4.3	Interpreter Subroutines Following the Regimen	7-11

tiny-c OWNER'S MANUAL
Version 1.00

BIBLIOGRAPHY	8-1
Appendix A -- Source Listings of the 8080 Tiny-c Interpreter	A-1
Appendix B -- Source Listings of the PDP-11 Tiny-c Interpreter	B-1
Appendix C -- "Crunched" Program Preparation System Listing	C-1
INDEX	I-1

ACKNOWLEDGMENTS

We are indebted to many friends and associates. Their enthusiasm for this project gave us encouragement when we needed it, and their feedback helped us improve tiny-c, especially to clarify rough parts of early drafts of this manual.

Rolf Levenbach, Rudy Libenschek, and Vic Ransom were pioneer users of tiny-c, and their lunchtable feedback led to many improvements. Professor Lou Katz at Columbia University was the pioneer installer of an 8080 version. His students laboriously typed in the hex code for the interpreter, because we couldn't find a machine-readable media common to our two systems!

Both the tiny-c logo and our cover design were done by Elizabeth Meredith of Valley Graphics. The book itself was produced, edited, and indexed by Maria Nekam.

We are grateful for their contributions.

Thomas A. Gibson
Scott B. Guthery

May 1978

ACKNOWLEDGMENTS

We are indebted to many friends and associates. Their enthusiasm for this project gave us encouragement when we needed it, and their feedback helped us improve tiny-c, especially to clarify tough parts of early drafts of this manual.

Rolf Lavenbach, Rudy Libencher, and Vic Hanson were pioneer users of tiny-c, and their invaluable feedback led to many improvements. Professor Lou Katz at Columbia University was the pioneer installer of an 8080 version. His students laboriously typed in the hex code for the interpreter, because we couldn't find a machine-readable media common to our two systems!

Both the tiny-c logo and our cover design were done by Elizabeth Mandel of Valley Graphics. The book itself was produced, edited, and indexed by Maria Noham.

We are grateful for their contributions.

Thomas A. Gibson
Scott B. Guthery

May 1978

PREFACE

The sources of ideas that went into tiny-c are many. First there is BASIC [Kemeny & Kurtz 1967]. BASIC has become the de facto standard training language in the United States. It is popular in high schools, universities, even in industry, where it is used for some production work. Although BASIC has its faults, its one big strength is that it is easy to learn. This is largely because it offers a single computing environment. You can enter new program lines, change old ones, and start a program running all from one command environment. You do not have to remember the environment you are in, i.e., edit mode, compile mode, link mode, system mode, run mode, etc., when giving a command. There are no commands to shift from mode to mode. There are no relocatable object modules, link editors, and all the other paraphernalia of "real" computers. It is very simple and very adequate. Thus a focus is made on the essential elements of computing, as opposed to the elements of "wrestling" with a computer.

The LOGO language [Feurzeig 1975] is in many ways similar to tiny-c. It offers a well-structured language based on BASIC, as well as a single environment for programming and execution. LOGO was used experimentally in public schools with very young children. The experiment showed that children could grasp simple computer concepts and work through a prepared set of exercises, and then do creative work of their own.

C [Ritchie, Kernighan, & Lesk 1975] is a computer language designed by Dennis Ritchie, at Bell Telephone Laboratories. Tiny-c borrows its overall structure from C. C is broadly used in universities and in industry. It has been used to program a very advanced and powerful computer operating system, called UNIX® [Ritchie & Thompson 1974]. And yet it

® UNIX is a trademark of Bell Laboratories.

tiny-c OWNER'S MANUAL
Version 1.00

is a very simple language. C has no native input/output, e.g., read or print statements. Input/output is done using functions. Thus C concentrates on COMPUTING facilities, and allows external development or elaborations of input/output. Tiny-c has adopted this idea.

The command environment for tiny-c is written in tiny-c. It needs no translation to the micro-processor's machine language. This corresponds somewhat to the idea of using C as the programming language to implement UNIX. So, although intended as a training language for structured programming, tiny-c is a powerful language.

The tiny-c OWNER'S MANUAL is trying to reach four audiences at the same time. For those new to structured programming we have a brief tutorial and program walk-through so they can get the gist of it without getting bogged down in details. Experienced users of structured programming will find that the reference sections let them quickly discover the features of tiny-c. For those who want to know how the tiny-c interpreter works, we have described its operation. And, finally, for those who want to install tiny-c on their home computer, we have included a complete installation guide.

FOREWORD

C is a versatile, expressive general-purpose programming language which offers economy of expression, modern control flow and data structures, and a rich set of operators. C is not a "very high level" language, nor a "big" one, nor is it specialized to any particular area of application. But its absence of restrictions and its generality make it remarkably convenient and effective for a wide variety of computing tasks.

C is concise -- you don't have to write a lot to get a job done. Yet at the same time, C programs are readable -- you can understand what you (or someone else) have written. This combination of brevity and readability is rare in programming languages, and is part of the reason that C is so widely used.

With tiny-c, Tom Gibson and Scott Guthery have designed a stripped-down version of C that is well-adapted to the microcomputer environment. Tiny-c retains C's expressiveness, conciseness and readability, yet sacrifices very few of its features.

At the same time, tiny-c provides a computing environment that will make it easier to develop programs. It comes with an editor and other piece parts that together make a program preparation system.

The tiny-c OWNER'S MANUAL is more than a reference manual for tiny-c, however. It is also a vehicle for conveying ideas and insights about how to get the most out of your machine, and about good programming in general.

tiny-c OWNER'S MANUAL
Version 1.00

C has simply taken over in many computing environments, not because people have been ordered to use it, but because it is a good language. It seems very likely that tiny-c will have a similar effect in the microcomputer world.

Brian W. Kernighan
Bell Laboratories
Murray Hill, New Jersey

May 9, 1978

I. INTRODUCTION

What is tiny-c? Tiny-c is

- a language, plus
- a standard library, plus
- a program preparation system.

Without any other software aids, you can prepare tiny-c programs, run them, edit them, store them on a cassette or floppy disk, and read them back later.

Tiny-c is a structured programming language which has if-then-else, while-loops, functions, global and local variables, and character and integer data types, pointers, and arrays.

Tiny-c is independent of operating systems. You can interface it easily to the input/output routines on your computer.

Tiny-c can invoke your own machine language subroutines so the tiny-c programming language can be fitted to your system and your system can be reflected in and extend the language.

1.1 A Tiny-c Program Walk-Through

Figure 1-1 is a complete tiny-c program consisting of two functions.

FIGURE 1-1

```

/*guess a number between 1 and 100
/* T. A. Gibson, 11/29/76

guessnum [
    int guess, number
    number = random (1,100)
    pl "guess a number between 1 and 100"
    pl "type in your guess now"
    while (guess != number) [
        guess = gn
        if (guess == number) pl "right !!"
        if (guess > number) pl "too high"
        if (guess < number) pl "too low"
        pl ""; pl ""
    ] /* end of game loop
] /* end of program

/*
/* random-generates a random number

int seed, last /* globals used by random
random int little, big [
    int range
    if (seed == 0) seed = last = 99
    range = big - little + 1
    last = last * seed
    if (last < 0) last = -last
    return little + (last/8) % range
]

```

End of FIGURE 1-1

How does this program work? Let's do a program walk-through:

Starting at the top, the first two lines are COMMENTS. A comment starts with /* and goes to the end of the line.

"guessnum" is the name of a FUNCTION which is called to start the program.

Following "guessnum" is a COMPOUND STATEMENT, which is 12 lines long, the last line being:

```
]      /* end of program
```

A compound statement is everything between balanced left-right brackets.

The first SIMPLE STATEMENT in guessnum is:

```
int guess, number
```

This declares two integer variables named "guess" and "number". All variables in tiny-c must be declared. When executed, the int statement will create the variables, and give them an initial value of zero.

The second simple statement in guessnum is

```
number = random (1,100)
```

This sets number equal to the value of the tiny-c program function random executed with its first argument equal to 1 and its second argument equal to 100. In our program the function random returns a random number between 1 and 100.

On the next line, pl is a tiny-c library function which prints a line. It prints the quoted string which is its argument.

while sets up a loop. The general form of while is:

```
while (expression) statement
```

In this instance, the expression part is

```
guess != number
```

where != means not equal to. This expression is evaluated, and if it is true, the statement is done, and then the expression is evaluated again. If it is false, the statement is skipped. Initially, guess is 0 and number cannot be less than 1, so the expression is initially true. Therefore the statement is executed.

The statement is compound, and is composed of six simple statements. The first of these statements is

```
guess = gn
```

gn, which stands for "get number", is another standard library function. It reads a number typed in by the user at the terminal, and returns that value. So here the program waits until the user types a number and a carriage return, and then guess is assigned the number typed.

The next three simple statements are if statements. The general form of the if statement used here is

```
if (expression) statement
```

where statement is executed if expression is true.

Statements five and six of the while's compound statement are pl"". pl"" goes to a new line, and prints nothing. The semicolon allows you to write more than one simple statement on the same program line. So

```
pl"" ; pl""
```

prints two blank lines.

Now we are at the end of the while loop. Since the expression part of the while was true, the while statement is executed again. This starts with another evaluation of the expression to see if it is true or false. If the first guess is not equal to number, the compound statement is executed again. Another guess is read, the appropriate remark is made, and two more blank lines are printed; the while is done yet again. Eventually the user gets the right number and guess is equal to number. This will cause a "right!!" and two blank lines to be printed. The while condition is then tested again. The expression guess != number is evaluated and found to be false, so the entire compound statement part of the while is skipped, which brings us to the end of guessnum. The game is over. The program stops because the end (the last]) of guessnum is reached.

Before we walk through random, notice the integers seed and last are declared outside of both guessnum and random. They are called GLOBAL variables. They will be created once when the program is started. They are initially zero, and are

known and usable by both guessnum and random. On the other hand, guess, number and range are LOCAL VARIABLES. guess and number are known and usable only within guessnum, while range is local to random.

The first line of random gives the function name. And, before the [, it declares two integer ARGUMENTS, little and big. A value must be supplied for each argument when a function is called. The call in the sixth line of guessnum sets little to 1, and big to 100. Now we enter the BODY of the function random.

range is declared an integer and is initially zero. On the first call, seed is zero. Now seed and last are both set to 99. range is calculated, and is 100. last is set to the product of last and seed which is 9801. This is not less than 0, so the statement part of the if is not evaluated.

We next come to the return statement. It does two things. First, it evaluates the expression. The result is made the VALUE OF THE FUNCTION. Second, it returns control to the program that called the function. Tiny-c expressions are similar to algebraic expressions. The symbol + means add, / means divide, and - means subtract (or take the negative). To indicate multiplication, a * is used. An unusual symbol is %, which means divide the left side by the right side and take the REMAINDER (not the quotient). So, for example,

$$1225 \% 100$$

is 25.

Thus the return statement calculates the expression:

$$\begin{aligned} & \text{little} + (\text{last}/8) \% \text{range} \\ &= 1 + (9801/8) \text{ remainder } 100 \\ &= 1 + 1225 \text{ remainder } 100 \\ &= 1 + 25 \\ &= 26 \end{aligned}$$

The value 26 is returned as the value of function random. It also leaves 9801 in last, and 99 in seed. Since these are

global variables, their values are retained between function calls. This is not true of local variables like range. Their values are retained only during the execution of the function in which they are defined. When that function is left their values are lost.

On a second call to random, range is recreated, and reinitialized to zero. seed is not zero, so seed and last are not set to 99, but remain 99 and 9801 respectively. range is recalculated as 100. Then

```
last = last * seed
      = 9801 * 99
      = 970299
```

This number is too big for tiny-c. Any computer has a limit on the size of the numbers that can be computed. Tiny-c numbers must be in the range

$-32768 \leq \text{number} \leq 32767$.

last OVERFLOWS this range. It will be assigned the value -12741! (We explain this more completely in Section 2.11.) This is less than 0, so the next statement assigns last the value 12741. Then the return statement calculates:

```
1 + (12741/8) remainder 100
= 1 + 1592 remainder 100
= 93
```

This is returned as the second value of random.

REVIEW OF THE WALK-THROUGH

The purpose of the walk-through is to get a feeling for programming in tiny-c. We have seen that

- * A tiny-c program is a set of functions.
- * Some functions are standard library functions, like gn and pl.
- * Global variables stay around and hold their values. Local variables come and go.

- * Function and variable names can be as long as you want.
- * A group of statements enclosed in brackets makes a compound statement which can be treated just like a simple statement.

1.2 Structured Programming -- What Tiny-c Is All About

Perhaps you have heard structured programming described as "go-to-less" programming. Or programming with just if-then-else and do-while control statements. Such remarks oversimplify what structured programming is all about. The essence of structured programming is PROGRAM CLARITY. You can write programs in small, modular parts, with easy-to-follow program flow. You can use well-chosen, descriptive variable names. This leads to clear, understandable programs. Program clarity is what structured programming is all about.

We discuss here four principle ideas that make program clarity possible. These are: modularity, predictable program flow, local variables, and the simple idea of meaningful variable names.

MODULARITY in software is just as important as modularity in hardware. It makes it humanly possible to deal with complexity. A module is a brick or atom used for building bigger modules. Seen from within, a module may be very complex but from the outside it is an indivisible whole. Software modularity is achieved through the use of FUNCTIONS.

PROGRAM FLOW is predictable if you can point to any statement and easily answer the question "under what conditions is this statement executed?" This is particularly important if the program is 20 or 30 pages long, and still has bugs. Scanning the whole program and drawing arrows is no fair. That's not considered an easy way to answer the question. Predictable program flow can be achieved in many ways. In tiny-c, it is done with COMPOUND STATEMENTS.

Functions also make it possible to hide variables used in a strictly local context. The variable `n` is very popular; it's used frequently to count things. Have you ever had a program blow up because you were using `n` in two places for two purposes? The fix was to change one of them to `n1`. A better idea is in the concept of LOCAL and GLOBAL variables.

As for long, MEANINGFUL NAMES for variables and functions -- just look at the sample programs to see the improvement.

David Gries suggests structured programming be called "simplicity theory", and characterizes it as "an approach to understanding the complete programming process" [Gries 1974]. As a pleasant dividend, structured programming is more enjoyable than monolithic programming. It should certainly, therefore, be a part of personal computing. To begin our look at tiny-c as a structured programming language, let's look at the foundation of functions and predictable program flow -- the compound statement.

1.2.1 Compound Statements

When you write a program, you write a list of statements:

```
x = x-1
a = b+c
b = b*2-c
x = b-a
```

The idea behind a compound statement is to make one statement -- a molecule -- out of a set of statements -- some atoms. This is done in tiny-c by

```
[x = x-1
a = b+c
b = b*2-c
x = b-a]
```

Anywhere you can write a simple statement you can also write a compound statement. This sounds simple, but the effect is powerful. For example most programming languages have an if statement similar to this:

```
if (logical expression) statement
```


So you can write

```
if (x>0) x = x-1
```

But make the statement part compound, and you have this capability:

```
if (x>0) [  
    b = b*2-c  
    a = b+c  
    x = x-1  
]
```

This multiline if is not some special kind of if. It is still:

```
if (logical expression) statement
```

But the statement part is compound. The compound statement is treated as an indivisible unit. It is either all done or all not done depending on the value of the logical expression.

The compound statement also is a natural for loops. There is a big difference among the various programming languages in how you write loops, but they all have one thing in common. A loop has a beginning and an end. A compound statement can be used to express this. The looping statement is:

```
while (logical expression) statement
```

Notice the similarity with the if. Only the keyword has changed. Here's how while works. The logical expression is evaluated. If it is true, then the statement is executed, and then the while is done again. The effect is a repeated if, i.e., a loop. As long as the logical expression remains true, the statement is done again and again. Eventually something in the statement causes the logical expression to become false, and the loop terminates. Of course, the statement can be compound, as in:

```
while (x>0) [  
    a = b+c  
    b = b*2-c  
    x = x-1  
]
```


The compound statement is a natural way to delimit the beginning and end of loops.

With one simple idea, the compound statement, two things are achieved. The if statement is more powerful than is common in non-structured programming languages. The concept of a loop collapses to a simple repeated if or while statement. In both situations you are stating conditions under which the statement -- whether simple or compound -- is to be executed.

1.2.2 Nesting Compound Statements

ANYWHERE YOU CAN WRITE
A SIMPLE STATEMENT, YOU
CAN WRITE A COMPOUND
STATEMENT.

That is a fundamental rule. A compound statement contains simple statements. Therefore a compound statement can contain compound statements. Figure 1-2 illustrates this.

FIGURE 1-2

```
[ x = x-1
  a = b+c
  b = b*2-a
  x = b-a
]
```

a=b+c is a simple statement. The rule says a compound statement can be written here. For example:

```
[ x = x-1
  [ a=b+c
    w = y+2*x+w
    y = 17
  ]
  b = b*2-a
  x = b-a
]
```

End of FIGURE 1-2

The substitution of a compound for a simple shown in Figure 1-2 is certainly allowable, but is of no practical value.

The real utility in nested compounds is in writing nested if and while statements. Figure 1-3 is therefore a more realistic example of the use of compound statements.

FIGURE 1-3

```

if (x>0) [
    while (x<limit) [
        if (case==1) [
            y = 0; w = 99
        ]
        if (case==2) [
            y = 99; w = 0
        ]
        nextaction
        x = x+1
    ]
]

```

End of FIGURE 1-3

In Figure 1-3, if you remove everything except the brackets, you have this:

```
[ [ [ ] [ ] ] ]
```

This is what is meant by nested compound statements. Brackets are used to form program units the same way parentheses are used to create arithmetic statements. The main difference is that a pair of brackets is preceded by a function name, or a logical expression. In the first case you're naming the contents of the brackets and in the second you're stating the conditions under which the contents are to be executed.

1.2.3 Readable Program Flow

In Figure 1-3, look at the "y=0" in the fourth line. How can it be reached? Only if case is 1, and x is less than limit. No go-to can lead here, either accidentally or on purpose.

How can "nextaction" be reached? Only if x is less than limit, and then only after possible changes to y and w. This

program has simple, predictable flow. The only way a statement other than a while can be reached is from directly above. whiles can also be reached from their matching] below.

1.2.4 Indenting and the Placement of Brackets in Compound Statements

The brackets alone define the "structure" of a program. Indenting means nothing. But one of the purposes of structured programming is to make programs more readable and, hence, more understandable. A good choice of indenting style is very important to program readability. There are several styles to choose from. The actual choice is not too important. But once you choose a style, stick to it. Consistency IS important.

One easily explained style is to align matching brackets vertically. This looks like:

```
if (x<0)
[
    statement
    statement
    "
    "
    "
]
```

A problem with this is that when editing the first statement, care must be taken to keep the [intact. So some use this style:

```
if (x<0)
[
    statement
    statement
    "
    "
    "
]
```


This takes an extra line. Also there is a visual break between the if and its statements. So some take the left bracket and move it to the end of the preceding line:

```
if (x<0) [  
    statement  
    statement  
    "  
    "  
    "  
]
```

The right bracket is now vertically aligned with the if or while that preceded the compound statement.

You may pick one of these, or invent a style of your own. But, we repeat, whatever you decide to do, do it consistently.

1.2.5 Functions

A large software project can usually be broken into natural parts, and each part programmed and debugged as a separate unit. Each of these units then becomes a reliable building block for the construction of still larger parts of the project. Sometimes units can be designed to be useful in many projects.

In various programming languages these building blocks are called subprograms, subroutines, or, as in tiny-c, FUNCTIONS. Here is a tiny-c function for any computer versus human game:

```
game [  
    getready  
    while (stillplaying ()) [  
        humanturn  
        if (stillplaying ()) computerturn  
    ]  
    gameover  
]
```


The name of the function is "game". The compound statement that follows is called the body of the function. Each [can be read as "do all of this", and its matching] read as "end of this". game divides the design of a game program into five parts:

getready (which initializes things, and
prints instructions if
requested),

stillplaying (which determines if the
game is still going, and
returns true if it is, other-
wise false),

humanturn (which conducts the human's
turn),

computerturn (which conducts the com-
puter's turn),

gameover (which computes and prints
scores, makes remarks about
the human's skill, promotes
the human, or whatever).

The game function is the first step in divide-and-conquer or top-down program development. Let's carry this development one step further. The getready function can be expanded this way:

```
getready [  
    ps "Do you want instructions?"  
    if (gc()=='y') instructions  
    setupboard  
]
```

getready divides the initialization into two parts: instructions, and setupboard.

(Note: ps prints a character string, gc() reads a character, and == 'y' tests if the character is a y.)

Notice that both game and getready are universal. They can be used in many game programs. Programming in this fashion eventually leads to a library of useful, general purpose

functions. These can be pulled off the shelf into a software project. You know they work because they were used before. Your programming becomes more productive, and more pleasant.

The next time you're programming a sizable project, i.e., anything more than a page, try to identify subsets of the logic usable in other projects. Capture these as functions. It is gratifying to discover a general purpose function where none was suspected.

1.2.6 Local and Global Variables

A LOCAL VARIABLE is one that is known only inside a function. It can be used and changed only within the body of the function. Even its name is unknown outside the function. In fact, its name can be used in other functions without conflict. This is what makes local variables useful.

Take a look at Figure 1-4. There are four local variables in these two functions. The variables `n` and `maximum` are local to `afunction`. The variables `n` and `total` are local to `anotherfunction`. If either of these functions calls the other, the values of `n` will not be confused since they only have meaning inside the body of their own functions. It helps to think of local variable names as being preceded by the possessive form of the function to which they are local. For example, `afunction's n` and `anotherfunction's n`.

FIGURE 1-4

```

afunction [
    int n, maximum
    n=0
    while (n<maximum) [
        .
        .
        .
        n=n+1
    ]
]
anotherfunction [
    int n, total
    .
    .
    .
    n = n+2
    total = total+n
    .
    .
    .
]

```

End of FIGURE 1-4

The value of locals is obvious to anyone who has spent a nasty debugging session trying to find out where, in a huge program, some variable is getting changed.

Of course not all variables can be local. Some must be shared by many functions. These are called GLOBALS. They should be used infrequently, as they do cause debugging headaches. Choosing good, descriptive names for globals alleviates the problem. A global named "k" is inviting disaster. Call it "klingsleft" and you're less likely to accidentally use it for two purposes. Also you've given a reader of your program a pretty good clue to the variable's use.

1.2.7 Summary -- And Where We Go from Here

We've walked through a simple program to get a feel for tiny-c, and we've discussed the virtues of structured programming. These are just the preliminaries. Now it's time for the main events. First, a complete definition of the tiny-c language. Chapter II is devoted to this task. To prepare programs you need an editor and a way of debugging. The Program Preparation System (PPS) is described in Chapter III. Examples are excellent learning tools: Chapter IV has several sample programs. Maybe you want to make it bigger, better, or faster? Chapter V explains how tiny-c works. Finally, of course, you'll want to get tiny-c up and running on your own computer. Chapters VI and VII explain how to install tiny-c on an 8080 or PDP-11®.

II. THE LANGUAGE

The tiny-c programming language is described completely here.

2.1 Comments

Comments begin with /* and continue to the end of the line. Apostrophes ('), quotes ("), and brackets ([]) should not be used in comments.

2.2 Names

Names of functions and variables can be one or more characters long. If more than eight characters are used, only the first seven and the last are significant. The first character must be alphabetic, either upper or lower case. The rest must be alphanumeric. Names cannot have imbedded blanks. Upper and lower case are considered distinct, so

```
GUESS
guess
```

are different names.

Names may NOT begin with any of these:

```
if
else
while
return
break
char
int
MC
```


2.3 Data and Variables

There are two kinds of data, integers and characters. A DATUM is an actual value:

```
7 is an integer datum
'a' is a character datum
```

A VARIABLE is a named cell that holds one datum. A variable must be created by declaring its existence and the type of datum it can hold in an int or char statement. For example, the two tiny-c statements

```
int a,b
char letter
```

declare the existence of three variables; two integer variables named a and b and one character variable named letter. Each can hold one datum of its respective type. Initially, integers are 0 and characters are ASCII null, i.e., 0.

2.3.1 Arrays and Array Elements

An ARRAY is a list of variables of the same type.

```
int values (10)
```

declares an array with eleven integer elements.

An array element is picked out using a subscript. For example,

```
value(7)
```

names the seventh element of the array value. Every array has a zero-th element

```
value(0)
```

and a last element

```
value(10).
```


When you declare an array, you name its last element. Thus, value has eleven elements:

```
value(0), value(1), ..., value(10)
```

The subscript of an array can be any expression.

```
value(i + 10 * k)
```

Even in a declaration, the subscript can be an expression. This is a convenient way of setting several arrays to the same or related sizes.

```
int size
size = 10
int x(size), y(size), matrix(size*size)
```

Note that matrix is NOT a two-dimensional array, but a single list of 101 elements. However, it can be addressed as a two-dimensional (0-9, 0-9) matrix this way:

```
int row,col
row = 7
col = 3
matrix(row + size * col) = ...
```

2.3.2 Locals, Globals, and Arguments

There are three scopes of variable declarations.

Locals: Local variables are declared within the body of a function (i.e., inside the [] part of the function.)

Arguments: Function argument variables are declared after a function name, and before the [beginning the function body.

Globals: Global variables are declared outside all functions.

In Figure 1-1, guess, number and range are local variables. The first two, guess and number, are local to the function

guessnum. The last, range, is local to the function random. The variables little and big are arguments. The variables seed and last are globals as are the function names guessnum and random.

Locals are created when a function is entered, and destroyed when the function is exited. When they are created they are also set to 0, (ASCII null, for characters).

Function arguments are always copied into a function, and then treated as locals.

Global variables may be accessed by all functions and preserve their values between function calls.

Within a function, the following names can be used:

- locals for the function,
- arguments of the function,
- all globals for the program, and
- all functions for the program.

Technically, arguments are locals, and function names are globals, so this rule is easier to remember as:

A FUNCTION CAN USE
ITS OWN LOCALS, AND
ALL GLOBALS.

All the locals within a function must have different names. But two different functions can each have their own local variable named x, and the two x's are kept separate.

All global names including function names must be different.

Duplicate names are not detected or diagnosed. A program will execute and ignore the second declaration of the name. The first declared name is always used.

One form of duplication is important. You can have a local and global variable of the same name. They are kept separate. Within the function that has the local, the local name prevails. Elsewhere, the global prevails.

2.4 Expressions

Expressions are formed from operators, parentheses, and primaries. They are used to calculate and store data, and to invoke functions.

2.4.1 Primaries

Primaries designate the data and/or destinations for results of expressions. They are the atomic elements of expressions. There are six types of primaries:

primaries =====	examples =====
constants	10, 'c'
strings	"hello"
variables	x
subscripted variables	buff(7)
array names	buff
functions	gn, ps "hello"

An integer constant may be signed. Its value must be between -32768 and 32767, inclusive. An integer uses two bytes of storage.

A character constant is always enclosed in apostrophes. A character uses one byte of storage.

Integers and characters are completely interchangeable in expressions. A character variable may be used as a one-byte integer whose value is in the range -128 to 127. This is occasionally useful, as in:


```
char newline, ch, digit
newline = 10      /* Puts an LF in new line.
ch = getchar
digit = ch - '0'  /* Converts an ASCII digit to
                  /* its integer value.
```

A character string constant is technically a two-byte constant which has as its value the address of the first element of an array of characters. Thus,

```
"hello"
```

is the address of the first element of an array of six characters initialized with h-e-l-l-o-null. Two-byte constants or variables which may also be used as addresses are called pointers. Thus, a character string is a pointer to its first character. Pointers are covered in detail later.

A subscript expression may be an arbitrary expression. The smallest subscript is 0, the largest is the declared size of the array. If an array's subscript falls outside these bounds, a subscript error is recognized. An exception to this rule is when an array is declared with size 0. Then any positive or negative subscript may be used. In effect, such an array can access any element in the direct address space of the computer.

Since function names have values, they may be used in expressions as though they were variable names. The value of a function name is the value returned by the function program of that name. In Figure 1-1, the use of the function name random

```
number = random(1,100)
```

is an example of the use of a function name as a variable.

2.4.2 Operators

The tiny-c operators are used to do arithmetic, compare values, and assign values to variables. We first show their

use in simple circumstances using one or two primaries. Then we consider more complex uses.

OPERATOR =====	USE ===	DEFINITIONS =====
unary +	+a	When used alone to the left of a variable, the plus sign is called a unary plus. It has no effect, and is used sometimes for readability.
unary -	-a	When used alone to the left of a variable, the minus sign is called a unary minus. The value of -a is the negative of a.
*	a*b	Multiplication. The value is a multiplied by b.
/	a/b	Division. The value is a divided by b. If there is a fraction, it is discarded, so the result is an integer. So 7/2 is 3. Also -7/2 is -3.
%	a%b	Remainder. The value is the remainder of a/b. So 7%2 is 1. Also -7%2 is -1.
+	a+b	Addition. Also called plus or binary plus. The value is the sum of a and b.
-	a-b	Subtraction. Also called minus or binary minus. The value is the difference between a and b.
<	a<b	Less than. The value is 1 if a is arithmetically less than b. Otherwise it is 0.
>	a>b	Greater than. The value is 1 if a is arithmetically greater than b. Otherwise it is 0.

OPERATOR =====	USE ===	DEFINITIONS =====
<=	a <= b	Less than or equal to. The value is 1 if a is less than or equal to b. Otherwise it is 0.
>=	a >= b	Greater than or equal to. The value is 1 if a is greater than or equal to b. Otherwise it is 0.
==	a == b	Equal to. The value is 1 if a and b are equal. Otherwise it is 0.
!=	a != b	Not equal to. The value is 1 if a and b are not equal. Otherwise it is 0.
=	a=b	Assignment. a is assigned the value b. Then the expression a=b assumes the value of b.

USES OF ASSIGNMENT: The assignment operator = can be used anywhere a binary + or - can be used. For example,

$$x(k=k+1) = a-(b=c/d)$$

performs three assignments. b is set to the value of c/d. k is set to k+1. The array element x (new value of k) is set to a minus new value of b. Also consider

$$a = b = c = 0$$

c is set to 0. Then b is set to c, i.e., to 0. Then a is set to b, also 0.

ORDER OF EVALUATION: When you write expressions with 3 or more primaries, the order of evaluation becomes important. For example, is

$$9 + 6/3$$

equal to 5 or 11? Standard algebra conventions would do the division first, then the addition. So the answer is 11. Tiny-c works the same way. All the operators are assigned a PRECEDENCE. In the absence of parentheses, the operators with the highest precedence are done first.

precedence =====	operators =====
highest	unary + unary - * / % + - < > <= >= == !=
lowest	=

So the value of

$7+3<5$ is $(7+3)<5$ is 0 [not 8].

$-1+7$ is $(-1)+7$ is 6 [not -8].

$a = 1+c = 2$ is $a = (1+c) = 2$ is illegal,
since you may not set an expression,
 $1+c$, to anything.

But

$a = 1+(c=2)$ sets c to 2 and a to 3.

$a = 1+c == 2$ is $a = ((1+c) == 2)$ which
sets a to 1 if $1+c$ is 2; otherwise
sets a to 0.

All the above cases are resolved by the precedence rule, but sometimes that is not enough. For example, is

$7 - 2 + 1$

equal to 4 or 6? Standard algebra would say 6. Note that $+$ and $-$ have the same precedence, so we cannot use precedence to determine which goes

first. The tiny-c convention is that the evaluation is done left to right. So,

$$7-2+1 = (7-2)+1 = 6.$$

$$7/2*2 = (7/2)*2 = 6. \text{ [Remember } 7/2 \text{ is } 3.]$$

But what about

$$a = b = c = 0$$

This is the exception. A series of assignments are done right to left.

USE OF PARENTHESES: Parentheses are used to change the order of performing operations. So in our very first example, if the desired result was 5, you can write it

$$(9+6)/3$$

An expression within parentheses is evaluated and then this value is used with operators outside the parentheses. Within parentheses, precedence and grouping rules determine order. So

$$22/(9+6/3) \text{ is } 22/11 \text{ is } 2 \text{ [not } 4]$$

because the precedence rule says $9 + 6/3$ is 11.

2.5 Functions

A function can be used as a primary anywhere in an expression (except to the left of an assignment.) When a function is used in an expression, we say the function is CALLED. When you call a function, it must be defined somewhere in your program, be in the standard library, or be the special function MC.

Every function is defined with a specific number of arguments. random has two arguments, little and big. When a function is called, values must be supplied for each of the

function's arguments. If you supply too many or too few values, an arguments error is recognized. Thus, for example, random must always be called with two arguments, ps must always be called with one, and gn must always be called with none.

The argument values are written as a list of expressions separated by commas. The list, even if empty, can always be enclosed in parentheses, and sometimes must be. Arguments themselves may invoke functions. Arguments are evaluated left to right. Here are some legal calls on random.

```
random (1, 100)
random (gn (), gn())
random (k = random (-10,10), k+10)
```

Notice each call to random has two arguments, because random is defined with two arguments.

If an argument is an array, then the supplied value must be a pointer of the same data type. For example, the argument to the library function pl is a character array. So a character pointer is the only valid argument.

```
pl "hello"
```

is valid, because "hello" is a character pointer to a string initialized with h-e-l-l-o-null. Other cases of pointer values are described in Section 2.6.

If an argument is not an array, then ANY value can be supplied. But usually it will not be a pointer. Neither argument to random is an array. So the supplied values may be of any type.

```
random 1,100
```

returns a number between 1 and 100.

```
random 'a','z'
```

returns a random lower case letter.

```
char a(100)
random a,a+100
```


returns a random address within a. Of course, this rule also applies to function definitions which are included in a tiny-c program. In the following example, the function len has one array argument and one non-array argument:

```

char a (10)
int k,l
k = len (a,l) /* a is a pointer
.
.
.
/* definition of len function
len
char string (10) /* string is an array
int n          /* n is not an array
[
.
.
.

```

Parentheses around the argument list are always allowed. Tiny-c allows them to be removed in certain cases. This is done principally to make input/output functions more legible. In the following forms, omitting parentheses around arguments is allowed.

```

k = function arg, arg, arg
k = function
function arg, arg, arg
function

```

The general rule is:

IF A FUNCTION AND ITS ARGUMENTS ARE THE LAST PART OF AN EXPRESSION, ITS PARENTHESES MAY BE OMITTED.
--

Whenever the function is involved in a more complex expression, parentheses must be used. For example

```
number = gn
```

is allowed, but `gn` in the expression

```
number = (gn () + 10)/2
```

needs the parentheses as shown.

There is no problem with complicated function arguments without parentheses. So this

```
pn 7 + 11/w - 142/g - x/17 + x%42
```

will print a number.

If the first argument begins with `(`, the argument list must be enclosed in parentheses. For example:

```
pn (7+2)/3
```

will do this:

- a) determine that `(7+2)` is the argument to `pn`,
- b) call `pn`, which prints a 9,
- c) `pn` returns a 0 as its value,
- d) `0/3` is evaluated, and the result discarded.

This is probably not what was desired. To print the value of `7+2` divided by 3 you should write

```
pn ((7+2)/3)
```

When a function is invoked with arguments, the value of each argument is passed to the function. A local copy is made within the function. The function can change the local copy, but this will not change the original.

For example

```

x = 10
blast x
pn x
.
.
.
blast int x [
x = 9999
]

```

The pn will print a 10. blast changes its local copy of x but not the "original" one.

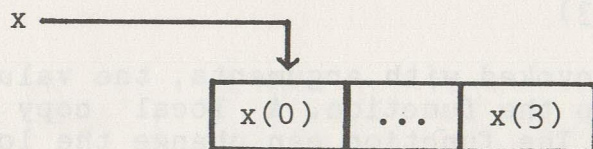
2.6 Pointers

A pointer is a memory address. A pointer variable is a variable whose value is an address. And a pointer expression is an expression whose value is an address.

We have seen that

```
char x(3)
```

declares a list of four character variables. They have names x(0), x(1), x(2), and x(3). In addition, it declares a pointer variable named x, and initializes it with the address of x(0). It is easy to visualize this way:



The arrow from x to x(0) indicates that the value in x is the address of x(0).

Pointers in Tiny C

if you declare a char array

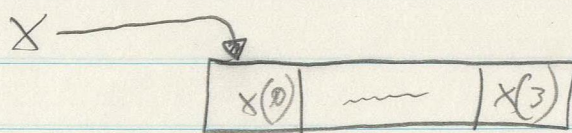
char x(3)

which is an array of 4 characters,

with names $x(0), \dots, x(3)$

then X is a pointer initialized

with the address of $x(0)$

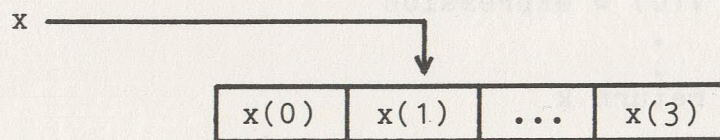


if you say $x = x + 1$, then x will point to $x(1)$

A pointer expression is a pointer plus or minus an integer expression. A simple case is $x+1$, which points to $x(1)$. A pointer variable can be assigned a new value. For example,

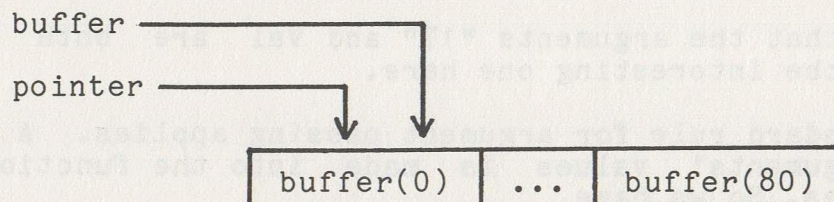
$x = x+1$

results in:

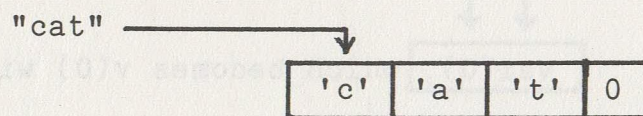


Or:

```
char buffer(80), pointer(0)
pointer = buffer
```



A character string is a pointer to an array initialized with the string and a null byte at the end:



Pointers are frequently used as arguments to functions because they let a called function change variables local to the CALLING function and thus return more than one result. The library function `num` is a good example. It must return both the number of characters scanned, and the value derived from those characters.


```

num char b(5);int v(0) [
    :
    :
    v(0) = 0
    :
    :
    v(0) = expression
    :
    :
    return k
]

```

The arguments to num are both pointers. Here is a call on num

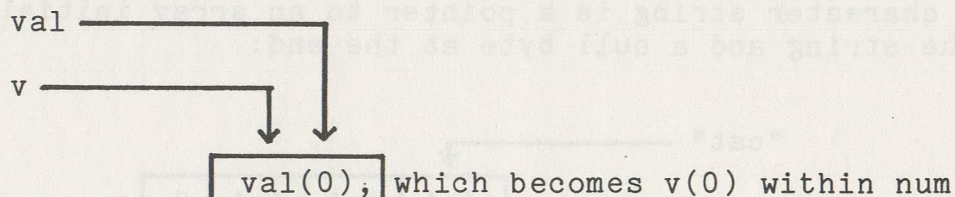
```

int val(0)
m = num "17", val

```

Notice that the arguments "17" and val are both pointers. val is the interesting one here.

The standard rule for argument passing applies. A copy of the arguments' values is made into the functions' local variables. So we have



The original argument, val, cannot be changed, but the word it points to can be. In fact, v(0) = 0 has the effect of changing val(0).

So the derived value is returned via the pointer v, and the number of characters scanned is returned as the value of num. In effect a call on num says "here are the characters to examine, and here is where to put the value of what you see".

There are other uses for pointers but this is a common one.

2.7 Statements

Simple statements end with a ;, or the end of the line, or the beginning of a remark. A left bracket, [, begins a compound statement. The matching right bracket,], closes it. A right bracket also ends the immediately preceding simple statement.

There are six tiny-c simple statements:

EXPRESSION

The expression is evaluated including all associated assignments and function calls.

if (EXPRESSION) STATEMENT

The statement is executed if and only if the value of the expression is non-zero. The statement can be compound, as in:

```
if (expression) [  
    statement  
    statement  
    "  
    "  
    "  
]
```

if (EXPRESSION) STATEMENT1 else STATEMENT2

Statement1 is executed if and only if the value of the expression is non-zero. Otherwise, statement2 is executed. Either may be compound. Whether compound or not, either can start on a new line. The else can also be on a separate line.

while (EXPRESSION) STATEMENT

The while statement works just like the if statement except the statement part is done repeatedly until the value of the expression becomes zero. The statement may be done as few as zero times, i.e. if the expression is initially zero. As with if and else, the statement may be compound, and can start on a new line.

return EXPRESSION

The expression is optional. If omitted, zero is used. The function containing the RETURN is exited. It is assigned the value of the expression.

break

The innermost while is terminated immediately, regardless of the value of its conditional expression.

2.8 Notes on Using the Statements

Section 2.7 defines the statements completely. Here are some not-so-obvious but frequently used consequences of their definitions.

if and while statements can be nested in any way. It is wise to use a consistent style of indenting when doing so.

Any expression can invoke functions, as described in Section 2.5. So the expression in an if, while, or return can invoke functions.

The statement in an if statement can also be another if statement as in

```
if (x<10) if (x>7) a = 1
```

Since the second if statement is a simple statement, no brackets are necessary. The variable a is set equal to 1 exactly when x is less than 10 AND x is greater than 7. Any number of expressions can be "anded" together like this.

The statement2 of an if-else can be another if-else, whose statement2 is another if-else, etc. For example:


```
command [
    char c
    c = gc
    if (c == 'p') print
    else if (c == 'd') delete
    else if (c == 'w') write
    else if (c == 'r') read
    else pl "must be p d w or r"
]
```

Each else follows its respective if. No nested brackets are needed. If, followed by a series of else if pairs, followed by an else, graphically displays an alternation. One and only one alternative is executed.

Always keep in mind that anywhere you can write a statement you can write either a compound statement or any of the six simple statements.

2.9 Libraries and Libraries and Libraries

The sample program in Figure 1-1 uses two functions from the STANDARD LIBRARY, pl and gn. It also uses random, from the OPTIONAL LIBRARY. In Section 1.2.5 we suggested you build a set of useful, general purpose functions, your own PERSONAL LIBRARY. In Section 2.10 we will describe Machine Calls, from which two additional libraries can be formed: STANDARD MACHINE CALLS, and PRIVATE MACHINE CALLS.

What is a library and what distinguishes one library from another? A library is simply a collection of similar functions, and libraries differ from one another on the basis of what the functions contained within them have in common. The five libraries that have been mentioned so far could be briefly defined as follows:

standard library -- tiny-c functions used by the PPS and useful to all programs developed with the PPS.

optional library -- tiny-c functions generally useful to tiny-c programs but not required frequently enough to be included in the standard library.

personal library -- tiny-c functions frequently used at a particular computer installation, or by a specific class of application program.

standard machine calls -- functions which are used so frequently by all tiny-c programs that they have been implemented in machine language to improve processing speed.

private machine calls -- functions which are so frequently used by tiny-c programs at a particular computer installation, or by a specific class of application programs that they have been implemented in machine language to improve processing speed.

The spirit of the standard, optional, and standard machine call libraries is that they have the same definition at each tiny-c installation so that programs developed at one can be run at another. They are, in a sense, extensions of the tiny-c language.

The STANDARD LIBRARY is a set of functions that is loaded with and used by PPS. As a result, these functions are accessible to all programs developed with PPS. They need not be defined or specifically loaded to be used. They are defined in Section 2.9.1 below.

The OPTIONAL LIBRARY is a set of tiny-c functions frequently useful to, but not always required by, a project. random is one of these functions. They are defined in Section 4.1. They are not loaded with PPS, so whenever they are used they must be specifically loaded. In principle, the optional library is a large collection of tiny-c program tools.

Your PERSONAL LIBRARY is your own extension to the optional library.

MACHINE CALLS are coded in machine language. There is a standard set furnished with tiny-c (Section 5.10); and you

can build your own private one (Section 2.10). These are used for speed, or to interface with special input/output devices.

Machine calls have an awkward, un-descriptive syntax. For example, it isn't immediately obvious what

```
MC 47,64,1,1001
```

means or does. It is customary to wrap a machine call up with a nice name, like this:

```
plot int r, c, nf [  
    return MC r, c, nf, 1001  
]
```

Now, when you write programs (especially for publication) you can use

```
plot 47,64,1
```

Although we haven't defined the plot function yet, "plot" is certainly somewhat more suggestive of what it does than "MC 1001".

2.9.1 Standard Library

The standard library includes functions that do input, output, and character manipulation. The definitions given here show the declaration of the function name and the arguments, if any.

```
gs char buffer(0)
```

Reads a line, i.e., a string of characters terminated by a carriage return, from the terminal and puts it in buffer. The carriage return at the end of the line is changed to a null byte. The value of the function is the number of characters placed in the buffer excluding the null byte. A value of 0 is permitted.

- ps** char buffer(0)
Prints the string in buffer on the terminal. A null byte signals the end of the string. The null is not transmitted. The number of characters transmitted is returned as the value of the function.
- pl** char buffer(0)
The same as ps but prints the string on a new line. The number of characters transmitted, not including the leading return and line feed, is returned.
- pn** int n
Prints on the terminal an integer preceded by a blank. The number of characters transmitted, including the blank, is returned.
- gn**
Reads a line, and returns the integer at the beginning of the line. If there is no integer there, it prints "number required" and tries again.
- gc**
Reads a line, and returns the first character on the line.
- putchar** char c
Transmits the character c to the terminal. Any character, including control characters, can be transmitted, except that if c is null a quote is transmitted. The character c is returned.
- getchar**
Reads and returns a character from the terminal. Any character, including control characters, can be read by this function.
- readfile** char name(0), where(0), limit(0)
int unit
Reads data from a file. name is a character string terminated by a null byte. where and limit are pointers. unit is an input/output unit (or device or channel). The file with name "name" is opened for reading on device "unit". All its records are read and placed in sequentially higher addresses starting at where, but in no case going beyond limit. Then unit is closed. If successful, the total number of bytes read is returned.

If limit is exceeded, the message "too big" is printed and -2 is returned. If any other problem occurs, installation-dependent messages may be printed, and a negative value is returned.

writefile char name(0), from(0), to(0)

int unit

Writes data to a file. name is a character string terminated by a null byte. from and to are pointers. unit is an input/output unit (or device or channel). writefile opens unit "unit" for output. The contents of sequentially higher addresses from "from" to "to" inclusive are written to unit as a file named "name". Then unit is closed. If successful, the total number of bytes written is returned. If a problem occurs, installation-dependent messages may be written, and a negative value is returned.

num char b(5)

int v(0)

Converts a string of digits without leading sign or blanks to the corresponding numeric value which is put in v(0). The first non-digit stops the conversion. At most, 5 digits are examined. The number of bytes converted is returned as the value of the function. Note that the second argument must be a pointer to an integer.

atoi char b(0)

int v(0)

Converts a character string of the form: 0 or more blanks, optional plus or minus sign, 0 or more blanks, 1 to 5 digits, to its numeric value which is put in v(0). The first non-digit following the digit part stops the conversion. The number of characters in b that were used to form the value is returned as the value of the function.

ceqn char a(0), b(0)

int n

Compares two character strings for equality for n characters. Returns 1 on equals, 0 on not equals.

alpha char c

Returns a 1 if c is an alphabetic character, upper or lower case. Otherwise returns a 0.


```
index char s1(0)
```

```
int n1
```

```
char s2(0)
```

```
int n2
```

Finds the leftmost copy ^{IN} ~~OF~~ the character string s1 which is n1 bytes long ^{IN} the character string s2 which is n2 bytes long. If s1 does not appear in s2, 0 is returned. If s1 does appear, return n+1 such that s2+n points to the first character of the copy in s2.

```
move char a(0), b(0)
```

Moves string a into b up to and including the null byte of a.

```
movebl char a(0), b(0)
```

```
int k
```

Moves a block of storage up or down k bytes in memory. a and b point to the first and last characters of the block to be moved. If k is positive the move is to higher addresses, and if it is negative the move is to lower addresses. If k is positive, the byte at b is moved first, then the byte at b-1, etc. If k is negative the byte at a is moved first. Thus large blocks can be moved a few bytes without destruction.

```
countch char a(0), b(0), c
```

Counts the instances of the character c in the block of storage from a to b inclusive and returns the count.

```
scann char from(0), to(0), c
```

```
int n(0)
```

Scans from "from" to "to" inclusive for instances of the character c. The integer n(0) is decremented for each c found. If n(0) reaches 0, or if the character in to(0) is examined, scann stops. scann returns the offset relative to the pointer, from, to the last examined character. Thus, if the third character position after from is the last examined, scann returns 3.

```
chrdy
```

Returns a copy of an input character from the terminal if a character has been typed but not yet read by another function, except that if the typed character is a null, a 1 is returned. If no unread character has been typed, a null byte is returned. The character is not cleared so a subsequent call to getchar or gc will return the same character.

pft char a(0), b(0)

Transfers all characters from a to b inclusive to the console terminal.

fopen int rw

char name(0)

int size, unit

Opens or creates a file for access on logical unit "unit". "name" contains a string, null terminated, giving the name of the file. There may be installation restrictions on file names. The file is opened for reading if rw is 1, and writing if rw is 2. If rw is 2 and the file does not exist, then it will be created and its size guaranteed to be at least "size" bytes. Otherwise size is ignored, but must be given. (Use a 0.) For a tape system, and some disk systems, rw and size may both be ignored, but they must be given nonetheless. If no error is detected, a 0 is returned. If an error is detected a nonzero is returned.

fread char a(0)

int unit

Starting at a, reads into memory the next record of data from the file opened on "unit". The array a must be large enough to hold the largest expected record. The length in bytes of the record is returned as the value of fread. Note that the installation may place an upper bound on record lengths. A -1 is returned if an end-of-file is detected, i.e., if an attempt is made to read beyond the last record in the file. A larger negative number is returned if an error is detected.

fwrite char from(0), to (0)

int unit

Writes one record with the bytes from "from" to "to" inclusive to the file opened on unit "unit". This becomes the next record of the file. Its length is from-to+1, and this is the length that will be returned when the record is read by fread. Note that the installation may place an upper bound on record lengths.

fclose int unit

The file opened on "unit" is made permanent, and arrangements are made for end-of-file detection by fread.

2.9.2 Notes on Using the Standard Library Functions

Note that the standard library has three functions to read characters. `getchar` and `gc` each read one character. `getchar` is the fundamental version. It:

1. Stops the program.
2. Waits for ONE CHARACTER to be typed.
3. Starts up again.
4. Returns the character.

`getchar` is useful for one letter commands. `gc` is oriented toward the input of full lines. It:

1. Stops the program.
2. Waits for a WHOLE LINE to be typed (ending with a carriage return).
3. Starts up again.
4. Returns the first character of the line.
5. Throws the rest of the line away.

`gc` is for one-letter answers to questions in a question and answer dialogue. A very common use is to test for the 'y' of a yes answer:

```
ps "Do you want to play again?"
if (gc() == 'y') [
```

```
  .
  .
  .
```

The third function, `gs`, reads a character string. Since `gs` is defined as

```
gs char buffer(0)
```

it must be called as

```
gs pointer-expression-to-character-data
```

For example,

```
char x(80)
gs x
```

will read a character string from the console terminal into `x(0)`, `x(1)`, ... If you wrote


```
gs x+10
```

the string would be read into x(10), x(11), ... Note that a quoted string is really a pointer. That's why

```
ps "hello"
```

works. The last quote of the string is replaced by a null within tiny-c.

Pointers are also used in standard library functions to return two or more results. num is an example. It returns the number of characters scanned. But it changes the integer v(0). Thus, num must be called like this:

```
int v(0), k
k = num "17", v
```

There is no subscript on v in the call. This call will put 17 into v(0), and 2 into k. atoi and scann also use this method.

atoi is just like num, except it handles a sign and leading blanks. num will NOT skip leading blanks.

ceqn is the standard way to compare two character strings since this type of comparison cannot be done with a tiny-c expression. For example, the following will NOT determine if "cat" has been entered at the console:

```
char x(10)
gs x
if (x == "cat") ... /*WRONG
```

The if will compare the pointer x with the address where "cat" is stored, and will always be false. To test if "cat" has been entered, use ceqn:

```
char x(10)
gs x
if (ceqn(x,"cat",3)) ... /*RIGHT
```

See Chapter IV for another match function, ceq.

movebl, countch, and scann are implemented in assembly language, and are, therefore, quite fast. index is implemented partly in assembly language and partly in

tiny-c. `scann` is intended to scan quickly for the `n`th occurrence of a character in an array. Here's how to use it.

```
int n(0),
char x(100), where
n(0) = 7
where = scann(x,x+100,' ',n)
```

This call to `scann` scans for the 7th blank in `x`. `x+where` will point to the 7th, and `n(0)` will be zero if there are at least 7 blanks. `where` will be 100 (the address relative to `x` of the last character examined) and `n(0)` will be 7 minus the number of blanks in `x` if there are less than 7 blanks. So testing `n` is a convenient way to see why `scann` stopped.

There are two facilities for manipulating data files. The simplest are the `readfile` and `writefile` functions. Whole arrays are read or written as whole files. Slightly more complex, but more versatile, are the `fopen`, `fread`, `fwrite`, `fclose` functions. These can access large files a record at a time.

For either facility unit 1 is guaranteed to be available on all installations. Other units are available only on multiple-unit installations. Note that a unit is a logical concept. For example, accessing unit 1 does not mean accessing drive 1 of a multidrive disk system. How units map onto devices is determined by the installation. But generally they will be small positive integers, e.g., units 1 through 4 on a four-unit installation.

Note that this specification leaves much to the installation. Tiny-c does not check that any of these limits are exceeded. Nothing is said in the specification about what happens when limits are exceeded. So if you write more than "size" bytes to a newly created file, you may abort, have your writes ignored, clobber an adjacent file, or your file may have its size extended and the writes will actually "take". It depends on the installation.

There are two points-of-view to take on exceeding the defined limitations of this package:

PRIVATE VIEW: Learn what your installation does when a limit is exceeded. If it is reasonable and useful, use the capability. But don't publish the program without warning others of what you have done.

PORTABILITY VIEW: Don't exceed the limits. Then you have a portable program that runs on anybody's installation of tiny-c.

Both views are valid, depending on your objectives. Clearly, PPS has adopted the portability view.

Among other limits is what happens if from-to+1 exceeds the installation record limit. As of this writing one installation truncates the excess, and writes one full record, whereas another writes multiple records so that all the data gets written. Each is reasonable and useful in the private view. In the portability view both installations do the same thing. In Section 4.3 the function writefile guarantees a limit of 256 bytes per record. So it is portable. Other "limits" are what happens if you read and write records to the same file, open the same file on two units, open the same unit on two files, etc. When adopting the portability view, don't do any of these things.

2.10 Machine Language Interface

There are several reasons why you may want to use your own machine language code for parts of a project. Usually this is done for execution speed, or to access new devices. If you write a machine language subroutine, and want to call it from a tiny-c program, the mechanism for doing so is the machine-call (MC) function. MC is passed arguments and returns a value. An argument can be an arbitrary expression. So, for example, you can write:

```
k = MC row-1, 2*col+6, 1, 1001
```

This passes four arguments to MC. The returned value is assigned to k. An MC can be used in an if statement, or an expression, or anyplace a function can be used:

```
if (MC(12)=='x') gotcha (MC(2))
```

This calls MC with the argument 12. If it returns the value 'x', then MC is called with the argument 2. The result returned is used as the argument in a call to gotcha.

The MC function differs from other tiny-c functions in three ways:

1. Its name, MC, is built into tiny-c.
2. It can have a variable number of arguments, but must have at least one. (Other functions must have exactly the number of arguments specified in their definitions, and can have none.)
3. It is coded in machine language, not in tiny-c.

The LAST argument determines which particular machine-coded function is to be executed. This argument is called the FUNCTION NUMBER.

Every function is assigned a unique number. Those furnished as "standard" MCs have numbers from 1 to 999. Those you write for your own local use can be assigned numbers from 1000 to 32767. This number assignment system guarantees that a future release of tiny-c won't have new function numbers that conflict with your own local ones.

An MC is invoked by tiny-c as follows:

1. The arguments are evaluated left to right, and their values pushed on a tiny-c stack. (This is not the processor stack, but a software implemented stack with special features.)
2. The last argument is the function number. It is popped from the tiny-c stack and examined. If it is less than 1000, the appropriate "standard" MC is executed.
3. If the function number exceeds 999, then 1000 is subtracted from it and a subroutine call is made to USERMC in the installation vector. You must place a jump instruction there to your MC code.

Note: In the 8080 version, this number is left in HL. In the PDP-11 version it is at 2(SP).

Now your MC has control. There are rules and tools for writing an MC:

1. You must write code to examine the adjusted function number and branch to your appropriate function. When done, a return is used to return control to tiny-c.
2. You must use all the arguments given.
3. You must return a result.
4. You cannot modify any standard cell used by tiny-c in its internal operation.
5. You must arrange for your MC code to be loaded, and the jump at USERMC must have the address where it receives control.
6. You must also arrange the four tiny-c data areas so they do not conflict with where you loaded your MC code. (See Section 6.5.2.)

Step 1 -- is easily done with 8080 code like this:

```

USERMC      JMP      MYMCS
             .
             .
             .
MYMCS        MOV      A,L           ;branch to one of
             CPI      1           ;two user MCs.
             JZ       MC1001
             CPI      2
             JZ       MC1002
MCERR        JMP      MCESET

```

Jumping to MCESET signals to tiny-c that an error was detected in an MC, in this case an invalid function number. MCESET is one of the MC writing tools. It can be used for other MC-detected errors.

In PDP-11 code the beginning of user MCs might look like this:

```

USERMC      JMP      MYMCS
            .
            .
MYMCS      CMP      #1,2(SP)
            BEQ      MC1001
            CMP      #2,2(SP)
            BEQ      MC1002
            MOV      #MCERR,-(SP)
            JSR      PC,@#ESET
            TST      (SP)+
            RTS      PC

```

Step 2 (8080) -- You can "use" an argument by calling the subroutine TOPTOI. (WARNING: This will modify all registers!) The value on the top of the stack is returned in DE, or just E if it is a one-byte value, and the stack is popped. Calling TOPTOI three times retrieves three arguments. Note that they are retrieved RIGHT TO LEFT as they appear in the MC call. Note also that the function number was already retrieved (popped) within tiny-c, so the first call gets the next-to-last argument. Don't call TOPTOI too often. There is no way to reassemble the stack the way it was, should you do so. Don't call it too few times. This leaves garbage on the stack, and the rest of your tiny-c program will get truly sick. What will probably happen is an arguments error for some innocent tiny-c function called later on. To help, the byte called MCARGS contains the number of arguments given by the call to the MC, including the function number. You can use this for checking, or for an MC with a variable number of arguments.

Step 2 (PDP-11) -- You can "use" an argument by calling TOPTOI. The value on the top of the stack is returned in R0 and the stack is popped. The arguments are retrieved RIGHT to LEFT as they appear in the MC call. The total number of arguments in the MC call is in 4(SP).

Step 3 (8080) -- To return a result, put a two-byte value in DE, and call PUSHK. This must be done ONCE in an MC before returning. Failure will probably cause an arguments error as described in Step 2.

Step 3 (PDP-11) -- To return a result, put the value to be returned in R0 and execute the following code:

```
MOV    R0,(SP)
MOV    #1,-(SP)
MOV    #101,-(SP)
CLR    -(SP)
JSR    PC,@#PUSH
ADD    #6,SP
```

This must be done ONCE in an MC before returning.

Step 4 -- Well, it's obvious you can cream tiny-c from a machine-coded subroutine. Just be careful.

Steps 5 and 6 -- How you assemble and load your code depends on your operating system. Be sure there is no conflict with other uses of memory. Section 6.5.2 describes how memory is allocated for tiny-c and includes recommendations for the placement of user MC code. Study this, and make adjustments to your memory allocation addresses so that your MC machine code doesn't overlap a tiny-c data area. Here's where some helpful jump addresses are located in 8080 tiny-c:

USERMC	ORG + 1F
MYMCS	determined by where User Machine Call program is loaded.
MCESET	ORG + 2B
TOPTOI	ORG + 2E
PUSHK	ORG + 31
MCARGS	ORG + 34

Section 6.2.2 defines ORG. The offsets are in hex.

Those are the basic steps to follow. Sample MCs (the 14 built-ins) are given in the listings and definitions are given in Section 5.10. These can be used as examples for building your own MCs.

2.11 Computer Arithmetic

All computers have limits on how large a number can be handled. When the limits are exceeded the number is said to OVERFLOW.

For both the 8080 and the PDP-11 versions of tiny-c, the numbers must be in the range

$$-32768 \leq \text{number} \leq 32767.$$

When a calculation would be outside this range, this is how to determine what happens. Subtract (or add if the calculation is too negative) 65536 repeatedly from the result until it does lie in the correct range. That is the answer.

The example in Section 1.1 is

```
last = last * seed
      = 9801 * 99
      = 970299
```

which is outside the range. Subtracting 65536 fifteen times gives the "correct" (sic) answer, i.e. the one returned by the computer:

$$\begin{aligned} &= 970299 - 15 * 65536 \\ &= -12741 \end{aligned}$$

III. THE PROGRAM PREPARATION SYSTEM (PPS)

The Program Preparation System (PPS) is a tiny-c program that lets you type in a program, edit it, run it, write it on a cassette or floppy disk, and read it back later for more runs and/or edits.

3.1 Fundamentals of PPS

A PPS session begins by reading PPS and starting it. (Refer to the installation chapter for your particular system to find out how to accomplish this.) PPS prints:

>

indicating it is ready for input. You can now type lines of your program, or give commands for PPS to execute.

Any line you type must end with a carriage return; PPS does nothing with your input line until the carriage return is given.

After the carriage return, PPS will either enter the line into your program, or execute the command. Then it gives another

>

and awaits another line or command.

If you mistype before giving a carriage return, the ASCII DEL character "kills" the most recently typed character. You can enter it several times to kill several characters. [Note: if DEL is unsuitable for your terminal, or your editing habits, the appropriate installation chapter describes how to modify this to a character of your choice.]

If you mistype a line so hopelessly that you want to do the whole line over, then the ASCII CAN in tiny-c/8080 or NAK in tiny-c/11 "kills" the whole line. CAN is control-X on most keyboards, while NAK is control-U. It must be given before the carriage return. You CANNOT give it several times to kill several lines. It will kill only the line which you have started, for which you have not yet given a carriage return.

If you have typed a carriage return and still want to make a change, this can be done. You must explicitly delete the line, using the delete command described in Section 3.3. Then you can re-enter the line.

DO NOT type in line numbers at the beginning of each line. Tiny-c does not use them, in fact, does not even tolerate them.

To indent, just type spaces. Do not use tabs. Tiny-c doesn't allow tabs, either.

Now, what is a command, and what is a line of text?

A COMMAND ALWAYS BEGINS
WITH A PERIOD, A PLUS, OR
A MINUS. ANYTHING ELSE
IS A TEXT LINE.

We will cover the PPS commands later; first, we will go over text lines. To do this we need the concepts of PROGRAM BUFFER, LINE ZERO, and CURRENT LINE.

As you enter program lines, they go into a character array called the PROGRAM BUFFER. Think of the program buffer as a series of text lines. If you enter a text line, it goes into the program buffer. It may be either added at the end of, or inserted between, lines already in the buffer.

Initially the program buffer has only one line, called LINE ZERO. Line zero has no text, just a carriage return. No matter what else you put in the program buffer, line zero is always there, and is always just a carriage return.

One line in the program buffer is always the CURRENT LINE. Initially it is line zero. You can always display the

current line by giving the print command:

>.p

(The ">" is the prompt printed by PPS. The ".p" is the print command typed in by the user.)

3.2 Entering Text Lines

Now, where do new text lines go?

A TEXT LINE IS ENTERED
AFTER THE CURRENT LINE.
THE NEWLY ENTERED LINE
BECOMES THE CURRENT LINE.

Initially the zero line is current. You type a text line. It goes into the program buffer as line 1, and becomes the current line. You type a second text line. It goes in after the current line (i.e., line 1), and becomes line 2; now line 2 is current. So if all you do is enter text lines, they each go into the program buffer one after the other.

There are commands (described below) to make current any line in the buffer. Whenever you enter a series of text lines, each is inserted one after the other below the current line. So you can enter text lines anywhere in the program buffer. You will discover that this text-line-entry rule is simple, natural, and powerful.

3.3 The PPS Commands

In the commands below, the "n" represents an unsigned (no + or -) integer. Exactly one blank must separate an integer n from preceding characters.

>.p Print the current line.

>.p n Print n lines, starting with the current line.
 The last line printed becomes current.

- >.d Delete the current line. Make the line BEFORE it current.
- >.d n Delete n lines, starting with the current line. Make the line BEFORE the first deleted line current.
- >+ Move down one line; i.e., make the line after the "present" current line, the "new" current line.
- >+n Move down n lines.
- >- Move up one line.
- >-n Move up n lines.
- >.n Make the n-th line in the program buffer the current line.
- >.l text Starting with the line AFTER the current line, and proceeding to the end of the program buffer if necessary, locate a line containing "text". If found, print the line, and make it current. If not found, print "?", and leave the current line unchanged. There must be one blank between the "l" and the first character of text. A "^" (ASCII octal code 136) as the first character of text means the text must begin the line. A "^" as the last character of text means the text must end the line. Text may contain blanks.
- >.l Same as above, but using the same text as given in a previous locate or change command.
- >.c text newtext

In the current line, the first occurrence of text is replaced by newtext. If text does not occur in the current line, no change is made. In either case the resulting line is printed. As shown, there are exactly two blanks in the command, one after the c, and one between text and newtext. In this form, no blanks can be used in text or newtext, because a blank is the delimiter that separates them. However, since any punctuation character can be used as the delimiter, the command can be given as:

>.c/text/newtext

In this case, blanks can be used in the texts, but /'s may not be used. An optional delimiter is permitted at the end of newtext:

>.c/text/newtext/

Either text or newtext can be empty.

>.c//newtext/

will insert newtext at the beginning of the line, whereas:

>.c/text//

will erase text from the line. Finally, when making a series of identical changes, you need not retype the texts over and over:

>.c Makes a change on the current line using text given in the most recent change or locate command, and newtext given in the most recent change command. The carriage return must be immediately after the c.

>./ Prints the current line number, the total number of lines, the total number of characters used in the program buffer, and the total number of characters unused.

>.r filename

Reads a file from cassette or floppy disk, putting what is read AFTER the last line. The current line is not changed. [Note: Some installations of tiny-c may not use the filename.]

>.w filename

Writes all lines to a cassette or disk, giving the name "filename" to what is written. [Note: Some installations may not use the filename.]

A command line starting with a period and at least two alphabetic characters is executed immediately as a tiny-c statement. This is how programs are started. Thus, to run the "guessnum" program in Figure 1-1:

```
>.guessnum
```

Arguments can be given. To add 7 and 11 and print the answer, call the pn library function:

```
>.pn 7+11
```

A compound statement can also be given, but it must fit on one line (64 characters including the period and the carriage return.)

```
>.[char a; while ((a=a+1)<=127) putchar c ]
```

Machine calls can be directly executed:

```
>.MC 24,64,1,1001
```

When a program is running, it can be halted and control returned to PPS by typing the ASCII ESC key. (Note: if ESC is unsuitable for your terminal, it can be changed to another character. See Section 6.5.4.2).

3.4 Notes on Using PPS

3.4.1 Bumping the Top and Bottom

Several commands can "bump into" the top or bottom of the program buffer. This is all right, and in fact, can be useful. For example, suppose there are 50 or so lines in the buffer. Then

```
>.0
```

```
>.p 999
```

makes line zero current, then prints the whole buffer. The

.p 999 "bumps into" the bottom of the buffer, and stops. The commands .d, +, -, .n, and .l can also bump into the top or bottom. A convenient way to go to the last line is:

>.999

3.4.2 Deleting

Line zero cannot be deleted. To delete lines at the top, go to line 1, then give the appropriate delete.

Notice that delete moves the current line up, not down. Thus any new lines typed after a delete will replace the deleted lines.

3.4.3 Line Numbers

When lines are inserted or deleted, lines further down in the buffer immediately have different line numbers. So .n is not the best way to locate a line. For example, in Figure 1-1, the command:

>.22

makes the first line of random current. But if edits are made to guessnum, then the 22nd line may or may not be the first line of random. For this reason, locate is a more powerful tool.

3.4.4 Using Locate and Change

The ^ convention in locate text makes it easy to locate the beginning of a function. To locate the random function:

>.1 ^random

A match occurs only if the text "random" is at the left margin of the page. So in Figure 1-1, line 6 will not match, but line 22 will match.

Locating all lines with a given text is done like this:

```
>.0  
>.1 random
```

This will match line 6 in the sample program. Then:

```
>.1
```

matches line 19, the comment containing the word "random". The following .ls match line 19, line 21, and line 22, while the final .l prints "?" indicating no further appearance of "random".

Making a common change throughout the buffer is just as easy. To change the variable named "number" to one named "num", type:

```
>.0  
>.1 number
```

This will make line 1 current. It is a comment so we do not want to change this occurrence of "number". Type:

```
>.1
```

and line 5 becomes current. We do want to change "number" here, so type:

```
>.c number num
```

Line 5 is changed. Continue with:

```
>.1
```

which makes line 6 current. Change it by typing:

```
>.c
```

and resume with:

```
>.1
```

You continue in this fashion, changing lines selectively, until you bump the bottom.

3.5 Errors

When a program error is detected, the program halts. A return is made to the system. Your program text is intact, and you can edit it, or restart it, or write it to a cassette. It prints three lines, as shown:

```
17  -- err 26
text of bad line
<
```

The first line shows the line number, and the error number. The second line is the text of the bad line. Immediately above or to the left of the < is where the problem was DETECTED. This may or may not be the real problem, depending on the logic of the program.

Upon halting, the current line is the line printed.

The error numbers and their meanings are:

- 1 Illegal statement
- 2 Cursor ran off end of program. Look for missing] or)
- 3 Symbol error. A name was expected. For example 10 + + will cause this.
- 5 Right parenthesis missing, as in: $x = (x+a*b$
- 6 Subscript out of range
- 7 Using a pointer as a variable or vice versa
- 9 More expression expected, as in: $x = x +$
- 14 Illegal equal sign, as in: $7=2$
- 16 Stack overflow. Either an expression is too tough, or you are deeply nested in functions, or a recursion has gone too deep.
- 17 Too many active functions
- 18 Too many active variables
- 19 Too many active values. Values share space with program text. Crunch the program and this error may go away. (Remove remarks and unnecessary blanks, and shorten variable names.) Or settle for fewer features, or buy more memory.
- 20 Startup error. Caused by a "garbage" line outside of all [], i.e., where globals are declared. A missing [or] can cause this.
- 21 Number of arguments needed and number given don't agree
- 22 A function body must begin with [.
- 24 An illegal invocation of MC
- 26 Undefined symbol. Perhaps name is misspelled, or you need an int or char statement for it, or the function isn't loaded.

3.6 Sample Session with PPS

```
>./
0 0 0
>/* Guess a number between 1 andq100
>.c/q/ /
/* Guess a number between 1 and 100
>/* T. A. Gibson, 11/29/76
>guessnum[
>  int guess, number
>  number=random(1,100)
>  pl "guess a number between 1 and 100"
>  pl "ttype in your guess now"
>  while(guess != number) [
>    guess = gn
>    if(guess == number)pl "right!"
>    if(guess > number)plq"too high"
>    if(guess < number)pl "too low"
>    pl"";pl""
>  ]      /* end of game loop
>]      /* end of program
>./
15 15 405
>.1
/* guess a number between 1 and 100
>.p 99
/* guess a number between 1 and 100
/* T. A. Gibson, 11/29/76
guessnum[
  int guess, number
  number=random(1,100)
  pl "guess a number between 1 and 100"
  pl "ttype in your guess now"
  while(guess != number)[
    guess = gn
    if(guess == number)pl "right!"
    if(guess > number)plq"too high"
    if(guess < number)pl "too low"
    pl"";pl""
  ]      /* end of game loop
]      /* end of program
```



```

>.r random
247
15 27 652
>.p
]          /* end of program
>+
/* random -- generates a random number between little
>.p 99
/* random -- generates a random number between little
/*          and big
random int little,big [
    int range
    if(last==0)last=seed=99
    range=big-little+1
    last=last*seed
    if(last<0)last=-last
    return little + (last/8)%range
]
int seed,last
>.guessnum

guess a number between 1 and 100
ttype in your guess now50

11 --- err 26
    if(guess > number)plq"too high"
    <
>.c/q/ /
    if(guess > number)pl "too high"
>.0

>.l yy
    pl "ttype in your guess now"
>.c/yy/y/
    pl "type in your guess now"
>.guessnum

guess a number between 1 and 100
type in your guess now50

too high

25

```


too low

37

too high

27

too high

26

right!

>.w guess

3 27 651

651

>

IV. TINY-C PROGRAM EXAMPLES

Since we all know the value of pictures and words, this chapter is devoted to tiny-c program examples. Section 4.1 contains some software tools which are candidates for inclusion in the optional library. Section 4.2 contains a complete original computer game called Piranha Fish. The tiny-c owner initially interacts most with the PPS; thus, Section 4.3 provides a readable and fully commented version of PPS together with the standard library. Sections 4.4, 4.5 and 4.6 show how tiny-c can be interfaced with specific hardware devices.

Programming can best be learned by reading programs. Such reading helps you learn style, idiomatic usages, and, in general, get an appreciation of the possibilities of a language.

The sample programs included here are intended not only to be useful, but also to be read. Therefore (wherever appropriate) we have commented on their style.

4.1 Optional Library Functions

These routines complement those in Section 2.9. We give their definitions, then the code, then a few comments on the programming style.

```
random int little, big
```

A pseudo-random number between little and big (inclusive) is generated. little cannot be larger than big, and their difference should not be larger than 4096. The global integers seed and last are part of random. These can be initialized to any value. Their value determines the sequence of numbers generated.

htoi char b(0)
int val(0)

Converts a string of hex digits to an integer. The hex digits may be preceded by blanks. The value of the first non-hex digit (except for leading blanks) stops the conversion. The integer is put in val(0), and the number of characters scanned in b, including blanks, is returned.

blanks char b(0)

Counts the number of leading blanks in the string b, and returns the count.

ceq char a(0), b(0)

Matches the two strings a and b up to but not including a null byte in a. 0 is returned on mismatch, 1 on match. Thus, a must be a leading substring of b to get a match.

char b(0)
ceq b, "yes"

returns 1 if b has "y", "ye", or "yes".

itoh int n

char b(4)

The integer n is converted to four hex digits plus a null byte, which are put in b.

itoa int n

char b(7)

The integer n is converted to an ASCII representation of the integer: a minus (-) if needed, followed by 1 to 5 digits followed by a null byte. The string is put into b. The number of bytes of the string (excluding the null) is returned.

moven char a(0), b(0)

int n

n bytes are moved from a to b.

4.1.1 Tiny-c Code for the Optional Library

random is shown in Section 1.1 (Figure 1-1). The other functions are given here.

FIGURE 4-1

```

/* Converts hex to integer. Returns result in val(0).
/* Returns number of characters scanned as value of htoi.
/* First non-hex character stops the scan.
htoi char b(0)
    int val(0) [
    int n /* Number of chars scanned.
    b=b+(n=blanks(b)) /* Skip blanks. Set b to first nonblank.
                        /* Set n to number of blanks skipped.

    val(0)=0
    while(1) [
        if(b(0)<'0')break
        else if(b(0)<='9')val(0)=16*val(0)+b(0)-'0'
        else if(b(0)<'A')break
        else if(b(0)<='F')val(0)=16*val(0)+b(0)-'7'
        else break
        b=b+1
        n=n+1
    ]
    return n
]
/* Counts leading blanks in b.
blanks char b(0) [
    int n
    while(b(n)==' ')n=n+1
    return n
]
/* Tests if a is a leading substring of b. Returns 1 on
/* true, 0 on false.
ceq char a(0), b(0) [
    while(a(0) != 0) [
        if(a(0) != b(0)) return 0
        a=a+1; b=b+1
    ]
    return 1
]
/* Converts integer to hex.
ith int n
    char b(4) [
    int k
    b(k=4)=0

```



```

while((k=k-1)>=0) [
    b(k)=n%16+'0'
    if(b(k)>'9')b(k)=b(k)+7
    n=n/16
]
]
/* Converts binary integer to ASCII.
itoa int n
    char b(7) [
        if(n<0) [
            b(0)='- '
            return 1+itoa(-n,b+1)
        ]
        if(n<10) [
            b(0)=n+'0'
            return 1
        ]
    ]
    int k
    b(k=itoa(n/10,b))=n%10+'0'
    b(k+1)=0
    return k+1
]
/* Move n bytes from a to b.
moven char a(0), b(0)
    int n [
        if(n)movebl(a,a+n-1,b-a)
    ]
]
>

```

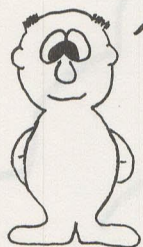
End of FIGURE 4-1

4.1.2 Comments on Style

Most of the code is straightforward. `ceq` is interesting because it increments the pointers `a` and `b`, instead of declaring an integer subscript and incrementing the subscript. Of course, only the local copy is being incremented. The pointers passed as arguments into `ceq` are not modified. This follows the general rules discussed in Sections 2.5 and 2.6.

`itoh` knows it's going to derive four hex digits. It gets the last one first, and puts it into `b(3)`, and then works towards `b(0)`.

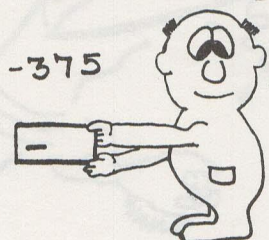
① Hi! I'm I to A, pronounced eye-to-A. I convert an integer to a ascii character. I recursively use myself to do my job.



And so I give my caller these 5 bytes:

-375null

② Give me -375 and a buffer (a character array). I put - in the first byte of the buffer and hand off 375 and the buffer offset by one...

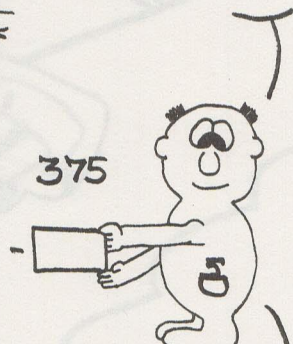


...byte to I to A to finish.

③ Of course I'm the only one that knows the buffer really starts one byte earlier and has a - in it. I put a null byte at the end.

④

⑤ Give me 375 and a buffer. I keep the 5 in my pocket, and hand off 37 and the buffer to I to A.

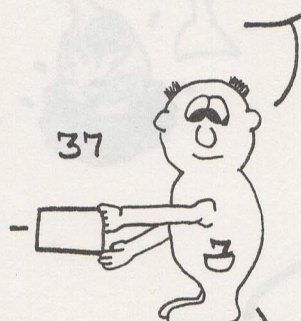


Now I stick my 5 at the end, and return 3 to signal that the buffer has 3 entries.

-375

⑥

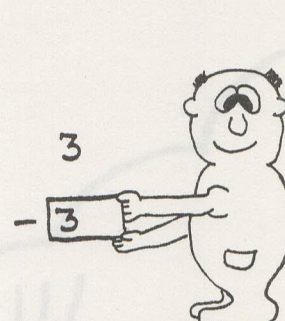
④ Give me 37 and a buffer. I keep the 7 in my pocket, and hand off 3 and the buffer to I to A.



Now I stick my 7 at the end, and return 2 to signal that the buffer has 2 entries.

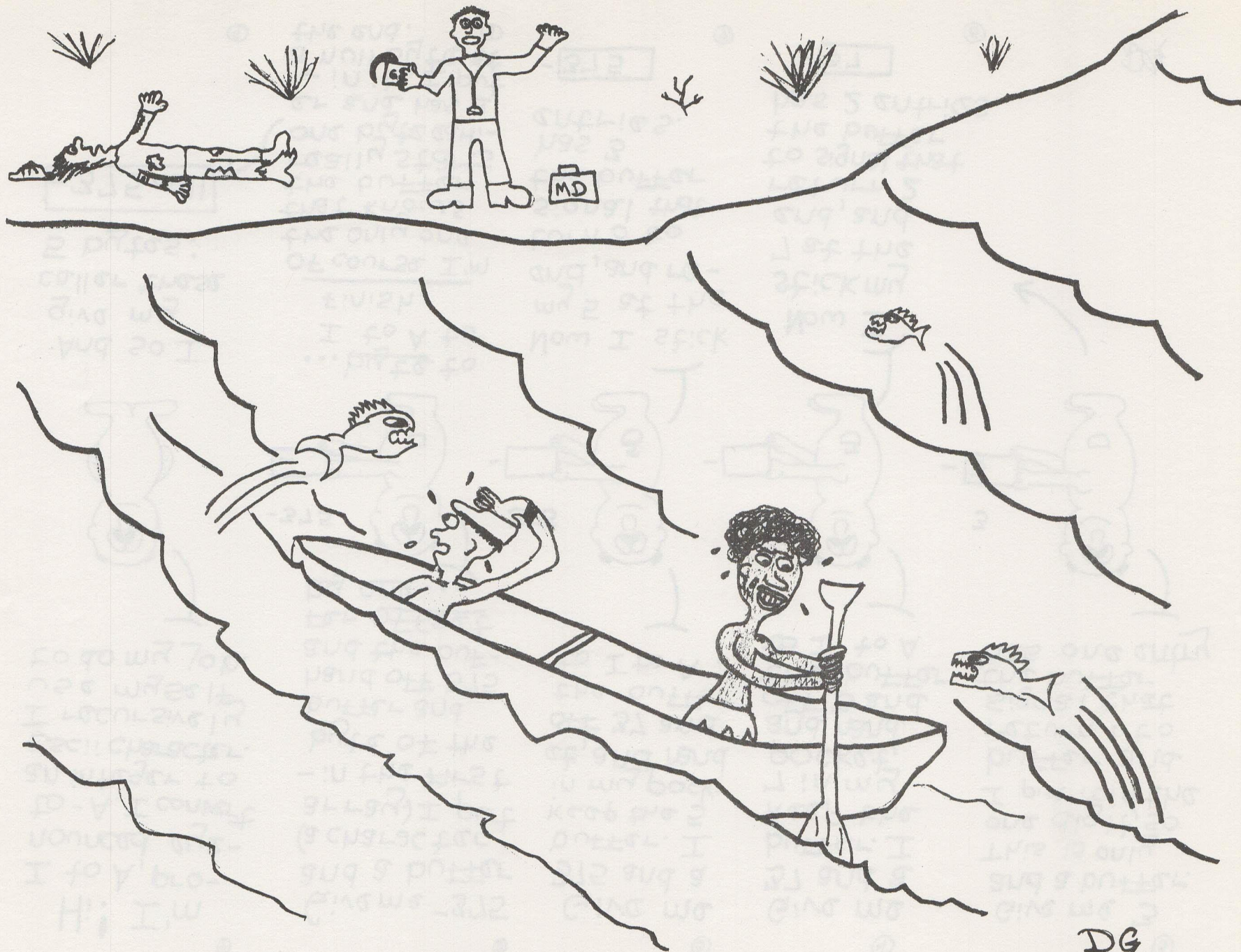
-37

③ Give me 3 and a buffer. This is only one digit, so I put it in the buffer and return 1 to signal that the buffer has one entry.



DL4

⑤



itoa also derives its last digit first, but it does not know in advance how long the string will be and hence where to put the first digit. There are several ways to handle this problem:

1. A series of if statements on n, e.g., $n < 9999$, $n < 999$, $n < 99$, $n < 9$, could be used to compute the size of the output string in advance. Then the technique for itoh can be used.
2. The bytes can be put into $b(0)$, $b(1)$, ... in that order, then reversed.
3. The bytes can be put into $b(6)$, $b(5)$, ... in that order, then moved left if needed.
4. Recursion can be used to get everything into place directly, with no size computation required.

itoa uses the last technique. It is a good example of recursion. The illustration on the facing page describes better than words how it works.

4.2 Piranha Fish -- An Original Game

You are leading the following party on a safari through the jungle:

- 2 cannibals
- 2 big-game hunters
- 1 doctor
- 1 nurse
- 3 missionaries

You arrive at a 100-yard-wide river filled with piranha fish. You must cross the river. There is a leaky canoe on your shore, which can hold, at most, 4 people. The cannibals paddle the best, followed by the hunters, the doctor, the nurse, and the missionaries, who are notoriously weak. You must decide who gets in the canoe for each trip back and forth. Get the party across with a minimum of carnage.

The doctor can attend major and minor wounds, unless he is himself wounded. The nurse can attend minor wounds. If the doctor is wounded, and the nurse is on the same shore as the doctor, she can (under his guidance) also attend major wounds.

Commands:

- s Prints status of game.
- digit For identification, each player is assigned a digit from 1 to 9. (See a status report). Typing a player's digit puts him in the canoe.
- Takes everybody out of the canoe. Use this when you have put somebody in, and change your mind.
- . Starts the trip.

Put all your commands on the same line. A carriage return is unnecessary. For example:

259.

puts players 2, 5, and 9 in the canoe, and starts the trip.

Now try a game or two. When you want to learn more, read facts. Good luck!

Type .pf to play the game. When "seed" is printed, enter a random number.

4.2.1 Facts

The speed of the canoe is the average of the paddling strengths of the players in the canoe. A speed of 100 gets the canoe to the opposite shore just as it fills.

The initial paddling strengths are:

cannibals	120
hunters	90
doctor	70
nurse	50
missionaries	40

Strengths are multiplied by the following factors for unhealthy paddlers:

minor wound, attended	0.9
major wound, attended	0.8
minor wound, unattended	0.8
major wound, unattended	0.7
dead	0.0

During a trip certain events happen, with probabilities shown:

Canoe fills at predetermined rate.

During each (speed/4) yard of the trip a single pf jumps in the boat with probability 0.25. He picks a random toe. Cannibals always spear the fish, and half the time make a hole in the boat. Hunters always panic, and capsize the boat. The doctor is quick half the time, and panics half the time. The nurse always panics; half of the time she is calmed down, and the other half, she jumps (alone) out of the boat and must swim ashore.

When the boat capsizes, everybody must swim. Dead players always float to the correct shore, and somehow the canoe gets there, too.

When swimming, the events that follow may occur to each player individually:

Dead players always float ashore.

Live players make it ashore unscathed half the time. The other half, they acquire minor wounds (prob. 0.67) or major wounds (prob 0.33). In no case do they come out of the river healthier than they went into it.

At the present time, these probabilities are independent of the length of the swim.
(Improvers take note.)

On the shore, a player's health can become worse:

Healthy players never get worse.

Attended players get worse with prob 0.11.

Unattended players get worse with prob 0.33.

Dead players never get worse.

To get worse means a minor wound becomes major,
or a player with a major wound dies.

When a minor attended wound gets worse, it
becomes a major unattended wound.

These "worse health" events are computed for
every player once per canoe trip, whether
or not the player participated in the
latest trip. So players wounded early
have more chances to get worse than
players wounded later.

Score:

1000 for a perfect game.

-100 per dead player.

-30 for major unattended wounds.

-15 for major attended wounds.

-10 for minor unattended wounds.

-5 for minor attended wounds.

Highest score achieved to date is 995.

Maximum Carnage Game

Certain people with twisted minds may decide to try for a minimum score. If you do this, a new rule is needed: On each successive round trip of the canoe, you must leave at least one more person on the far shore than on the previous round trip.

Happy paddling!

4.2.2 Piranha Fish Code

```
char shore(9),health(9),canoe,move(4),ngoing,afloat
int hfactor(6),sinkrate,paddle(9)
/* Conducts the game.
pf [
    setup
    while(stillplaying())[
        whosgoing
        trip
        shoreacts
    ]
    wrapup
]

/* Sets up initial conditions.
setup [
    hfactor(0)=10
    hfactor(1)=9
    hfactor(2)=hfactor(3)=8
    hfactor(4)=7
    paddle(1)=paddle(2)=12
    paddle(3)=paddle(4)=9
    paddle(5)=7
    paddle(6)=5
    paddle(7)=paddle(8)=paddle(9)=4
    sinkrate=25
    ps"seed"
    seed=last=gn
]

/* Game is still going if any player on shore 0 is alive.
stillplaying [
    int p
    while((p=p+1)<=9)
        if((shore(p)==0)*(health(p)<5)) return 1
]
```


2

```

/* Conducts dialog, determining which players make next trip.
whosgoing [
    char j,p,i
    char dup
    pl"";pl"move "
    while(1)[
        j=getchar
        if(j=='.'')[    /* Trip command.
            i=0
            while((i=i+1)<=ngoing)    /* At least one paddler required.
                if(health(move(i))<5)return
            ps" nobody to paddle "
        ]
        else if(j=='-')[    /* Unload command.
           ngoing=0
            ps" canoe emptied"; pl""
        ]
        else if(j=='s')[    /* Print board.
            status
            ps"move "
        ]
        else if((j>='1')*(j<='9'))[    /* Put player in canoe.
            p=j-'0'
            dup=0
            i=0
            while((i=i+1)<=ngoing) if(p==move(i)) dup=1
            if(dup) ps" already in boat "
            else if(shore(p)!=canoe) ps" on other shore "
            else if(ngoing>=4) ps" canoe full "
            else move(ngoing=ngoing+1)=p
        ]
    ]
]

```



```

/* status prints the board.
status [
    char k(0),p
    pl"";pl""
    ps "near shore                far shore  "
    pl"";pl""
    while((p=p+1)<=9)[
        if(shore(p)) ps "
        pn p; ps" "; pname p; ps" "
        if(health(p))[
            k="minor att  major att  minor unattmajor unattdead  "
            k=k+11*(health(p)-1)
            pft k,k+10
        ]
        pl""
    ]
    if(canoe)ps "
    ps "      canoe"
    pl"      "
    if(canoe)ps "
    char i
    while((i=i+1)<=ngoing)pn move(i)
    pl""
]

/* Conducts a trip across the river.
trip [
    char i
    int speed,dist,full
    afloat=1
    while((i=i+1)<=ngoing)
        speed = speed + paddle(move(i))*hfactor(health(move(i)))
    speed=speed/(4*ngoing) /* Yards per unit of time.
    while((dist=dist+speed)<100)[
        full=full+sinkrate
        if(afloat*(full>100))[
            pl"The boat is swamped....."
            capsize

```



```

        break
    ]
    if(afloat)[
        pl"Canoe has"; pn 100-dist; ps" yards to go, and is"
        pn full; ps"% full"
        if(random(1,4)==1)onefish
    ]
]
i=0      /* The far shore is reached.
while((i=i+1)<=ngoing) shore(move(i))=1-shore(move(i))
canoe=1-canoes /* Swap shores of players in canoe, and canoe.
ngoing=0 /* Everybody out.
pl"trip to "
if(canoes)ps"far"; else ps"near"
ps" shore is complete."
]

/* A fish jumped in the boat. This is what happens.
onefish
[
    char p
    pl"A piranha fish has jumped into the boat. He is swimming"
    pl"around. He is looking at the toe of the "
    pname(p=move(random(1,ngoing)))
    ps"."
    if(health(p)>4) pl"Oh, well. He's dead anyway....."
    else if(p>6)[
        pl"The missionary is calm. He is staring back at the"
        pl"fish. The fish just jumped back into the river."
    ]
    else if(p<3)[
        pl"The cannibal has speared the fish. "
        if(random(0,1))[
            pl"Unfortunately he made a hole in the"
            pl"boat, increasing its sink rate 10%."
            sinkrate=sinkrate+sinkrate/10
        ]
    ]
]

```



```

else if(p<5)[
    pl"The hunter has panicked.  He is rocking the boat..."
    capsize
]
else if(p==5)[
    if(random(0,1))[
        pl"The doctor is quick.  He shoots the fish full of"
        pl"a drug."
    ]
    else[
        pl"The doctor has panicked.  He is rocking the boooooat!"
        capsize
    ]
]
else[
    pl"The nurse has panicked.  She is rocking the boat."
    pl"Everybody is yelling at her.  Yell - yell - yell."
    if(random(0,1))[
        pl"She is calm now, and sits down."
    ]
    else[
        pl"She falls out of the boat.  She is swimming."
        swim 6
    ]
]
]

/* Player p swims to shore.
swim char p [
    if(health(p)>4)[
        pl"Player"; pn p; ps" floats ashore."
    ]
    else if(random(0,1))[
        pl"Player"; pn p; ps" makes it."
    ]
    else[
        pl"BYTE!! BYTE!!  Player"; pn p

```



```

if(random(0,2))[
    if(health(p)==2)health(p)=4
    else if(health(p)<2)health(p)=3
]
else if(health(p)<4) health(p)=4
if(health(p)==3) ps" fortunately escapes with minor wounds"
else ps " major wounds acquired."
]
]

/* The canoe is capsized.
capsize [
    char p
    pl"CAPSIZE!!! Everybody swim FAST!! The fish are coming.."
    while((p=p+1)<=ngoing)swim move(p)
    afloat=0
]

/* When on shore, some players get mended.
shoreacts [
    char p
    while((p=p+1)<=9)[
        if(shore(p)==shore(5))[ /* Doctor with at most minor wounds can attend all
            if(health(5)<4) if(health(5)!=2) /* wounds.
                if((health(p)==3)+(health(p)==4)) [
                    health(p)=health(p)-2
                    pl""; pn p; ps " attended by doctor."
                ]
            ]
        if(shore(p)==shore(6))[
            if(health(6)<4) if(health(6)!=2) if(health(p)==3) [
                health(p)=1
                pl""; pn p; ps" attended by nurse."
            ]
        else if(health(p)==4) /* (And also major wounds with the doctor's advice.)
            if(shore(5)==shore(6))
                if(health(5)<5) [

```



```

        health(p)=2
        pl""; pn p; ps" attended by nurse"
    ]
]
if(health(p)==0)[ /* All done if healthy.
else if(random(0,2))[] /* All done for .67 of sick.
else if(health(p)<3)[ /* But some get sicker.
    if(random(0,2)==0)[
        if((health(p)=health(p)+1)==3) health(p)=5
        pl""; pn p; ps" is much worse"
        if(health(p)==5) ps", in fact dead."
    ]
]
else if(health(p)<5)[
    health(p)=health(p)+1
    pl""; pn p; ps" is much worse"
    if(health(p)==5)ps", in fact dead."
]
]
]

/* Computes score.
wrapup [
    int s,h,p
    s=1000 /* Perfect score.
    while((p=p+1)<=9)[
        h=health(p)
        if(h==5)s=s-100
        if(h==4)s=s-30
        if(h==3)s=s-15
        if(h==2)s=s-10
        if(h==1)s=s-5
    ]
    pl"";pl""
    status
    ps"Your score is"; pn s
]

```


88

```
/* Prints a player's name.
```

```
pname char p [
```

```
char k(0)
```

```
if(p<3)ps "cannibal"
```

```
else if(p<5)ps "hunter"
```

```
else if(p<6)ps "doctor"
```

```
else if(p<7)ps "nurse"
```

```
else ps "missionary"
```

```
]
```


4.2.3 Comments on Style

Notice how the functionality of this program makes it readable. You can find a feature quickly, and modify it with confidence that the house won't fall in.

Note the use of the character pointer `k` in `status`. It is used to compute for printing one of five possible health messages, depending on the health of player `p`. The function `pft` is used to print exactly 11 characters. Thus, from three lines of code any one of five messages is printed.

Piranha Fish uses only standard and optional library functions, and standard MCs. These are all furnished with `tiny-c`. So if you can get `tiny-c` up, you've got this program in the bag. If you use a plot function from your personal library, dramatic improvements to this game are possible.

4.3 The Standard Library and PPS

PPS is defined in Chapter III. We give its code here, including the standard library functions. This listing is in "human-readable" form, i.e., with comments, indenting, and long names. It's about 9000 bytes long. The machine-readable version of this program was "crunched" to about 4000 bytes. In general, programs should be written in a human-readable style, then a crunched version produced if the situation warrants it. This one clearly does.

4.3.1 Program Preparation System Code

```
/* Transmits c to the terminal. If c is null transmits ".
putchar char c [
    if(c==0)c='"'
    return MC c,1
]
/* Reads one character from the terminal.
getchar [
    return MC 2
]
/* Reads a line from the terminal. Implements character and line delete.
gs char b(0) [
    int l
    while((b(l)=MC(2))!=13) [ /* Do until carriage return.
        if(b(l)==24) [ /* line kill
            l=0; pl""
        ]
        else if(b(l)==127) [ /* char kill
            if(l>0)l=l-1
        ]
        else l=l+1
    ]
    b(l)=0 /* Put null at line's end.
    return l
]
/* Prints a string.
ps char b(0) [
    int l
    char c
    l=-1
    while((c=b(l=l+1))!=0)MC c,1
    return l
]
```

For IBM PC, implement an MC
which does a BIOS call
to LINE INPUT SVC


```

/* Goes to new line and prints a string.
pl char b(0) [
    MC 13,1
    ps b
]
/* Tests if a is alphabetic.
alpha char a [
    if((a>='a')*(a<='z'))return 1
    if((a>='A')*(a<='Z'))return 1
]
/* Converts numeric character string b to integer. Puts value in v(0). Returns
/* the number of bytes examined. First non-digit stops the scan.
num char b(5)
    int v(0) [
        int k
        v(0)=0
        while(k<5) [
            if((b(k)<'0')+(b(k)>'9'))return k
            v(0)=10*v(0)+b(k)-'0'
            k=k+1
        ]
        return k
    ]
]
/* Converts signed integer character string b to binary integer. Puts value in
/* v(0). Returns the number of bytes examined.
atoi char b(0)
    int v(0) [
        int k,s
        char c
        s=1
        c=b(0)
        while((c==' ')+(c=='-')+(c=='+')) [
            if(c=='-')s=-1
            c=b(k=k+1)
        ]
        k=k+num(b+k,v)
        v(0)=s*v(0)
        return k
    ]
]

```



```
/* Prints a signed integer.
```

```
pn int n [
```

```
MC ' ',1
```

```
MC n,14
```

```
]
```

```
/* Reads a line from the terminal. Gets a signed integer from the beginning of the
```

```
/* line, and returns its value. Insists on getting a number.
```

```
gn [
```

```
char b(20)
```

```
int v(0)
```

```
while(1) [
```

```
gs b
```

```
if(atoi b,v)return v(0)
```

```
ps "number required "
```

```
]
```

```
]
```

```
/* Compares first n bytes starting at a with first n starting at b. Returns 1 on
```

```
/* match, 0 on no match.
```

```
ceqn char a(0),b(0)
```

```
int n [
```

```
int k
```

```
k=-1
```

```
while((k=k+1)<n) if(a(k)!=b(k))return 0
```

```
return 1
```

```
]
```

```
/* Searches string in (of length lin) for the first occurrence of the string find
```

```
/* (of length lfind). Returns 0 on not found, n>0 on found, where n is the
```

```
/* offset from in-1 where find was found.
```

```
index char in(0)
```

```
int lin
```

```
char find(0)
```

```
int lfind [
```

```
if(lfind<=0)return 1 /* Null text always found.
```

```
if(lin<=0)return 0
```

```
int at,left(0)
```

```
while(at+lfind<=lin) [
```

```
left(0)=1
```

```
/* seann finds first char fast.
```

*change to do char by char input, allowing only
digits and a leading optional sign & backspace*


```

    at=at+1+scann(in+at,in+lin-lfind,find(0),left)
    if(left(0))return 0 /* If no first char, then fail.
    if(ceqn(in+at,find+1,lfind-1))return at /* If match, then succeed, else go
                                           /* back to get another first
                                           /* character.
]
]
/* Move string a to b. First null in a stops the move.
move char a(0),b(0) [
    int k
    k=-1
    while(a(k=k+1)!=0)b(k)=a(k)
    b(k)=0 /* Move null too.
    return k /* Number of bytes moved.
]
/* Read a line, return its first character.
gc [
    char f
    f=MC 2
    while(MC(2)!=13) []
    return f
]
/* Move the block a...b up or down n bytes.
movebl char a(0),b(0)
    int n [
        MC(a,b,n,7)
    ] /* In the block a...b count occurrences of character c.
countch char a(0),b(0),c [
    return MC(a,b,c,8)
] /* Scan the block a...b for the n(0)th occurrence of char c. Reduce n(0) for
/* every c found. Return pointer to last character examined.
scann char a(0),b(0),c
    int n(0) [
        return MC(a,b,c,n,9)
    ]

```

Handwritten notes:

- } implement as MC?* (next to the string move function)
- } implement in MC* (next to the block move/count/scan functions)


```

]    /* Read the file 'name' into 'where', but take care not to read into bytes
    /*      beyond 'limit'. Use unit to do the read.
readfile char name(0),where(0),limit(0)
    int unit [
int k,total
MC(1,name,0,unit,3)    /* Open
while(1) [
    k=MC(where,unit,4)    /* Read a block
    if(k== -1)return total    /* End of file
    if(k< -1)return k    /* Error
    total=total+k
    if((where=where+k)>limit) [
        pl"Too big"
        return -2
    ]
]
]
/* Write the block from...to as a file named 'name', using unit to do the writing.
writefile char name(0),from(0),to(0)
    int unit [
    int k,total,1
    MC(2,name,to-from+1,unit,3)    /* Open for writing.
    while(from<=to) [
        l=to-from
        if(l>255)l=255    /* Blocksize-1 for the installation
        k=MC(from,from+l,unit,5)    /* Write one block.
        if(k<0)return k    /* Error
        if(k>0)return -k    /* Also error, but must return negative.
        total=total+l+1
        from=from+l+1
    ]
    k=MC(unit,6)    /* Write end-of-file.
    if(k<0)return k    /* Error
    if(k>0)return -k    /* Error
    return total    /* No error, return amount written.
]
endlibrary

```



```

int err(0)    /* err returned by application program.
int cursor   /* Pointer into current line.
int lineno   /* Current line number
int progend   /* Pointer to last byte of program.
int lpr      /* length of pr
char line(64) /* Input line
char pr(7000) /* Program buffer
int lline    /* Length of input line
int lastline /* Number of lines in pr
char ltext(20) /* Locate text
char totext(40) /* Change to text
int len,tolen /* Lengths of ltext, totext
main [
    char c
    int val(1)
    lpr=5500
    pr(0)=13 /* Create empty line 0.
    lastline=cursor=lineno=progend=err(0)=ltext(0)=0
    while(1) [
        ps">"
        while((lline=gs(line))==0) [] /* Read non-empty line.
        c=line(0)
        if(c=='.') [
            if(num(line+1,val))goto(val) /* .number
            else if((line(2)==0) + (alpha(line(2))==0)) [
                c=line(1) /* One-letter commands
                if(c=='p')print
                else if(c=='d')dlines
                else if(c=='l')locline
                else if(c=='c')change
                else if(c=='?')facts
                else if(c=='r')progin
                else if(c=='w')progoto
                else if(c=='x')return
                else [ps"???";pl""]
            ]
        ]else start /* Multi-letter commands
        ]
        else if(c=='-')up /* Not . try + -
        else if(c=='+')down
        else insert /* Not . or + or -
    ]
]

```



```

/* Prints n lines, making the last current
plines int n [
    int fc,lc,val(0)
    val(0)=n
    fc=fchar /* First char to print.
    lineno=lineno+val(0)-1
    lc=cursor+scann(pr+cursor,pr+progend,13,val) /* Last char to print.
    cursor=lc
    lineno=lineno-val(0)
    MC pr+fc,pr+lc,13 /* This does the printing.
]
/* Returns pointer to first char of current line.
fchar [
    int k
    if((k=cursor)==0)return 0
    while(pr(k=k-1)!=13)if(k<0)break
    return k+1
]
/* Returns pointer to last char of current line.
lchar [
    int k
    k=cursor-1
    while(pr(k=k+1)!=13)if(k>=progend)break
    return k
]
/* Advances cursor to next line.
nl [
    if((cursor=lchar()+1)>progend) [
        cursor=progend
        return 0
    ]
    return lineno=lineno+1
]

```



```

/* Backs cursor to previous line.
bl [
    if((cursor=fchar()-1)<0)cursor=0
    else lineno=lineno-1
]
/* Prints a set of lines.
print [
    int val(0)
    if(line(2))num(line+3,val)
    else val(0)=1
    plines(val(0))
]
/* Deletes a set of lines.
dlines [
    int fc,lc,val(1)
    if(cursor==0) [
        ps"cannot delete line 0";pl""
        return
    ]
    if(line(2)==0)val(0)=1
    else num(line+3,val) /* val is how many lines to delete.
    lastline=lastline-val(0)
    fc=fchar /* First char to delete.
    lc=cursor+scann(pr+cursor,pr+progend,13,val) /* Last char to delete.
    lastline=lastline+val(0) /* In case val is too big, adjust lastline by the
    /* excess.
    lineno=lineno-1 /* Back up current line.
    cursor=fc-1
    if(lc<progend)movebl(pr+lc+1,pr+progend,-(lc-fc+1)) /* Closes the non-deleted
    /* text.
    progend=progend-(lc-fc+1)
]

```



```

/* Locates a line with given text.
locline [
    int k
    if(line(2)!=0) [
        len=move(line+3,ltext) /* Set up locate text.
        if(ltext(0)=='^')ltext(0)=13
        if(ltext(len-1)=='^')ltext(len-1)=13
    ]
    if(len==0) [
        pl"locate what?";pl""
        return
    ]
    if(nl()!=0) [ /* Scan starts at next line.
        if(k=index(pr+cursor-1,progend-cursor+2,ltext,len)) [ /* index does the
                                                                /* search.

            cursor=cursor-2+k /* Found, set cursor and lineno.
            if(pr(cursor)==13)cursor=cursor+1
            lineno=countch(pr,pr+cursor-1,13)
            plines 1 /* Print found line.
        ]
        else[ps"?"; pl""] /* Not found
    ]
    else[ps"at bottom"; pl""] /* No next line, can't even start.
]
/* Changes text within current line.
change[
    char del
    int ptr,fc,lc
    if(line(2)!=0) [ /* Default is previous locate, to texts.
        del=line(2) /* Delimits locate and to texts.
        ptr=2
/* Locate middle delimiter
        while((line(ptr=ptr+1)!=del) [
            if(line(ptr)==0) [ /* No middle delimiter
                line(ptr+1)=0 /* Second null creates empty to text.
                break
            ]
        ]
        line(ptr)=0 /* Null in middle delimiter.
        len=move(line+3,ltext) /* New locate text
        tolen=move(line+ptr+1,totext) /* New to text

```



```

        if(tolen)if(totext(tolen-1)==del)tolen=tolen-1    /* Remove optional third
                                                             /*      delimiter.
    ]
    fc=fchar    /* Scan line for ltext.
    lc=lchar()-1
    int k
    if(k=index(pr+fc,lc-fc+1,ltext,len)) [
        cursor=fc+k-1    /* Found, set cursor to where.
        movebl(pr+cursor+len,pr+progend,tolen-len)    /* Move tail end text in or out.
        progend=progend+tolen-len
        if(tolen)movebl(totext,totext+tolen-1,pr+cursor-totext)
    ]
    plines 1    /* Print the result.
]
/* Inserts one line of text after current line.
insert [
    lline=lline+1
    if(progend+lline>lpr) [
        ps"won't fit";pl""
        return
    ]
    if(nl)movebl(pr+cursor,pr+progend,lline)    /* Move tail out.
    else[cursor=cursor+1; lineno=lineno+1]
    progend=progend+lline
    movebl(line,line+lline-1,pr-line+cursor)    /* Move in new line.
    pr(cursor+lline-1)=13    /* Put return at end.
    lastline=lastline+1
]
/* Gives facts about current line, including err code.
whathappened [
    int fc,lc,lcurs,blanks
    pn lineno; ps" --- err "; pn err(0); pl""
    lcurs=cursor
    fc=fchar
    blanks=lcurs-fc
    lc=lchar
    fc=fc-1

```



```

while((fc=fc+1)<lc)putchar(pr(fc));pl""
while((blanks=blanks-1)>=0)putchar(' ')
putchar '<'; pl""
]
/* Moves cursor down.
down [
    int val(1)
    if(line(1)==0)val(0)=1
    else num(line+1,val)
    lineno=lineno+val(0)
    val(0)=val(0)+1
    cursor=cursor+scann(pr+cursor,pr+progend,13,val)
    lineno=lineno-val(0)
    plines(1)
]
/* Moves cursor up.
up [
    int lines,val(1)
    if(line(1)==0)lines=1
    else[ num(line+1,val); lines=val(0)]
    while((lines=lines-1)>=0)bl
    plines(1)
]
/* Move cursor to given line.
goto int l(1) [
    lineno=l(0)
    l(0)=l(0)+1
    cursor=scann(pr,pr+progend,13,l)
    lineno=lineno-l(0)
    plines(1)
]
/* Print some facts.
facts [
    pn lineno
    pn lastline
    pn progend
    pn lpr-progend
    pl""
]

```



```

/* Enter an application program.
start [
    while(lline<=64) [
        line(lline)=' '
        lline=lline+1
    ]
    MC(err,line+1,pr+progend,pr,11)
    if(cursor<0)cursor=0; if(cursor>progend)cursor=progend
    lineno=countch(pr,pr+cursor-1,13)
    pl"";pl""
    if(err(0))
        if(err(0)==99) [ps"stopped";pl""]
        else whathappened
    ]
/* Read a file into pr.
progin [
    int k
    k=readfile(line+3,pr+progend+1,pr+lpr,1)
    if(k<0)return
    progend=progend+k
    lastline=countch(pr+1,pr+progend,13)
    pn k; pl""; facts
]
/* Writes all of pr to a file.
progout [
    facts
    pn writefile(line+3,pr+1,pr+progend,1)
    pl""
]

```


4.3.2 Comments on Style

The library routines are clean, and fairly straightforward. index requires study to understand. The clue is this: to get a modicum of speed, at least the first byte had to be matched at machine code speed. scann had the ability to do this, so it was adopted. scann is a technically difficult function to master; this is a good demonstration of that fact. The problem was broken into two parts: match the first byte and then test if the remaining bytes match. The second part is done by the if(ceqn(...)).

movebl, countch, and scann (and a few others) are examples of wrapping an MC in a nice package, and tying a bow on it. This should be done to all MCs. Sometimes a modest variation can be made in the packaging, as in putchar, where nulls are mapped to quotes before MC 1 is called. This is done to keep the MCs "pure", while, at the same time, incorporating a small variation in its predominant usage. Another function could still use the MC without the variation, or with yet another variation.

Students of software archeology will recognize the PPS code as a product of evolution. Originally, the only way to move the current line was with the functions nl and bl. The up, down, and goto functions all calculated an appropriate number of nls or bls and then did them. They worked, but slowly. So goto was recoded using scann; in fact, it was the motivation for scann. Next, down was recoded using scann, making it fast also. But up still uses bl and is still slow. A cleaner design now would be to eliminate nl and bl altogether, and code up and down in terms of goto. But even with this dicotomy of method, the code is well structured. Most of the routines are quickly read and understood. locate and change are (not surprisingly) the only difficult ones.

4.3.3 The Use of MC 11

A tiny-c program loaded via the loader is, by definition, a LEVEL ZERO PROGRAM. Usually this is PPS, but it could be any system-type program. When a program uses MC 11 to invoke (not call) another program, the invoked program is given a level one higher than the invoking program. Thus, when PPS invokes an application, that application runs at level one. An application is also allowed to use MC 11, creating levels higher than one.

The principle need for this is in using PPS to program and test new or modified PPSs. A working PPS is loaded and started at level zero. An experimental PPS is loaded as a level one application. It is edited, started and tested just like any other application. When the level one experimental PPS is running, it is preparing the text of still another program, which, if started, runs at level two.

In tiny-c/8080, the global variable APPLVL contains the current application level. An application can be stopped by pressing the ESCAPE key. This terminates the program, and causes the invoking MC 11 to return. The application level is reduced by one. The error 99 is returned in FACTS. ESCAPE only works to terminate applications. It cannot terminate a level zero program. The choice of ASCII character that stops an application can be changed. The global variable ESCAPE contains the ASCII character that causes an application stop. It can be changed to any value not used in programming or for data (see Section 6.5.4.2).

Tiny-c/11 also contains a global variable called APPLVL which is incremented as MC 11 is entered, and decremented as MC 11 is left. This variable can be examined by interrupt routines to determine how to handle an interrupt.

4.4 Morse Code Generator

Do you have something on your computer that goes beep? For example, a printer that beeps when it's sent ASCII BELL? Some printers make a long continuous beep when sent several BELLS.

This program allows you to practice receiving individual letters in Morse code, or to send a Morse code message. Type

```
>.practice
```

to practice. Answer the four questions. (The last seeds the random number generator.) Then write down the letters beeped to you in Morse code. At the end, compare your answers with the ones displayed. Or have a friend type

```
>.morse "any message"
```

and listen to the message he is sending.

FIGURE 4-2

```

int SPEED
bell[
    MC 1002
    MC 7,1
    int k
    while((k=k+1)<3)[[]
    MC 1003
]
lots[
    while(1)bell
]
dot[
    bell
    pause
]
dash[
    bell; bell; bell; pause
]
pause[
    int k
    while((k=k+1)<SPEED)[[]
]
space[
    int r
    while((r=r+1)<20)[[]
]
letter char c [
    pause
    int k
    while((k=k+1)<SPEED)pause
    if(c=='a')[dot;dash]
    else if(c=='b')[dash;dot;dot;dot]
    else if(c=='c')[dash;dot;dash;dot]
    else if(c=='d')[dash;dot;dot]
    else if(c=='e')dot
    else if(c=='f')[dot;dot;dash;dot]
    else if(c=='g')[dash;dash;dot]
    else if(c=='h')[dot;dot;dot;dot]
    else if(c=='i')[dot;dot]
    else if(c=='j')[dot;dash;dash;dash]
    else if(c=='k')[dash;dot;dash]
    else if(c=='l')[dot;dash;dot;dot]
    else if(c=='m')[dash;dash]

```



```

else if(c=='n')[dash;dot]
else if(c=='o')[dash;dash;dash]
else if(c=='p')[dot;dash;dash;dot]
else if(c=='q')[dash;dash;dot;dash]
else if(c=='r')[dot;dash;dot]
else if(c=='s')[dot;dot;dot]
else if(c=='t')dash
else if(c=='u')[dot;dot;dash]
else if(c=='v')[dot;dot;dot;dash]
else if(c=='w')[dot;dash;dash]
else if(c=='x')[dash;dot;dot;dash]
else if(c=='y')[dash;dot;dash;dash]
else if(c=='z')[dash;dash;dot;dot]
else if(c==' ')[pause;pause;pause]
]
morse char s(0) [
    SPEED=12
    while(s(0))[
        letter s(0)
        s=s+1
    ]
]
practice [
    char c
    int k,n,r,r1
    ps"how many letters"
    k=gn
    ps"how many repeats"
    r=gn
    ps "speed"
    SPEED=gn
    ps "seed"
    seed=last=gn
    while((n=n+1)<=k)[
        c=random 'a','z'
        r1=0
        while((r1=r1+1)<=r)letter c
        letter ' '
        putchar c; putchar ' '
    ]
]

```

End of FIGURE 4-2

4.4.1 Comments on Style

You may have to replace dot and dash to conform to different hardware. You may also have to experiment awhile to get the timing correct. In bell, (which you may have to replace), two private MCs are used. 1002 enables the printer, and 1003 disables it. The MC 7,1 transmits ASCII BELL. Thus, nothing goes to the printer except BELLS. SPEED is set to a default value in line two of function morse. A lower value causes faster code generation.

4.5 A Tape-to-Printer Copy Utility

A handy utility is one that reads a file from a cassette or disk, and prints it. One would think that this is ordinarily an assembly language job. But here, written almost entirely in tiny-c, is the utility used to print Appendix A. The only parts not in tiny-c are two private MCs: 1002 enables the printer, and 1003 disables it.

The system for which this utility was written has a 300 baud printer and a 2400 baud cassette recorder. They both use the same USART, so only one device can be enabled at a time. This utility conforms to that restriction. It opens the file, reads one block of up to 256 bytes, and closes the file, again freeing the USART. Then it opens the printer, prints the block, and closes the printer, thus freeing the USART for use on the file.

FIGURE 4-3

```

/* Copies a file from cassette to printer.
pfile char name(0)[
  char a(256)
  int len,err
  while(1)[
    err=fopen(1,name,0,1)    /* Open to read a block.
    if(err)[
      pl"open err";pn err
      return
    ]
    len=fread(a,1)
    fclose(1)
    if(len<0)[
      if(len == -1) [pl"readblock err";pn len]
      return
    ]
    popen          /* Open the printer.
    pft(a,a+len-1)
    pclose
  ]
]
popen[MC 1002]
pclose[MC 1003]

```

End of FIGURE 4-3

4.6 TV Graphics Functions

There is a variety of devices available for plotting on a TV screen. Generally, they divide the screen into a rectangular grid and allow selective "painting" or "erasing" of any cell in the grid. Some provide for only black or white cells and some allow up to 16 colors. The tiny-c library does not include TV graphics functions because they are device-dependent.

Here are two function specifications for black and white TV graphics, but easily modified for color. We assume the device has R rows and C columns of cells. The rows are numbered top down from 0 to R-1, the columns left to right from 0 to C-1.

clean

The screen is erased.

plot int row, col, onoff

Tests if row and col designate an onscreen spot (i.e., $0 \leq \text{row} < R$, and $0 \leq \text{col} < C$). If this is NOT true, plot takes no action, and returns a 1. If it is true, and if onoff is nonzero, the spot designated by row and col is "painted" white or turned on; if onoff is zero, the spot is "painted" black or turned off. In either case, plot returns a 0.

The definition of plot can be extended to color grid cells by giving meanings to different nonzero values of onoff. Note also that this definition requires the plot function to accept ANY value for row and col, even one wildly off-screen. plot cannot abort or modify a random memory byte just because row and col are off the screen. It simply plots nothing and returns a 1.

The next program demonstrates the use of plot.

4.6.1 Meteor Shower

Meteor shower is a graphic display program consisting of a main program called "star" and a function called "shoot". star queries the user for a seed for random, and the number of fixed and of shooting stars. The first while statement puts down the field of fixed stars. The second while statement causes a series of shooting stars to cross the screen. Most of the work is done by the function shoot, which puts one shooting star across the screen. It chooses a random entry point along the top edge, but not too close to the corner. It finds its angle of descent by setting delta at random. delta is the amount of horizontal motion for each vertical step. delta ranges from -3 to +3, and it results in a size between 0 and 4. This is skewed to favor small sizes



DAD, ARE THERE
SUCH THINGS
AS PIRANHA FISH?

UH-OH! I
PUSHED
"DEL" BY
MISTAKE.

OH NO...

I GET
CONFUSED,

IS IT
I to A,
A to I,
OR BOTH?

WHAT'S A
CANNIBAL?

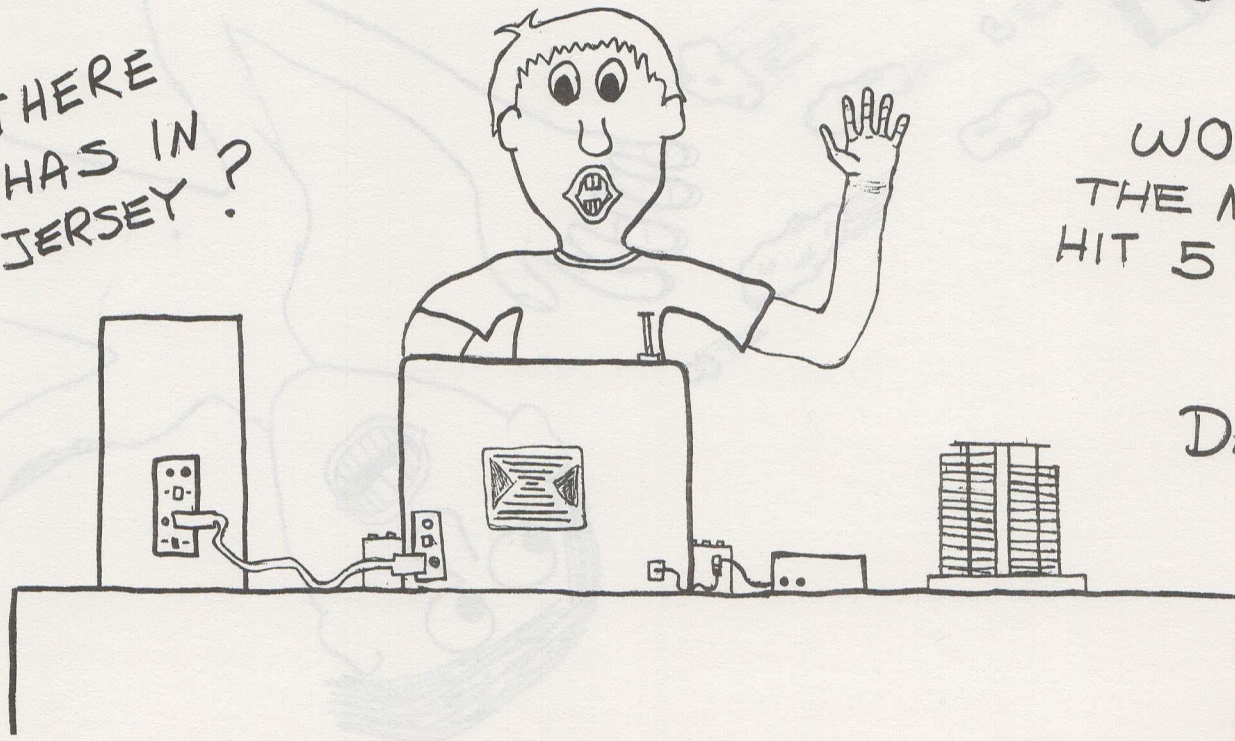
BEEP... BEEP!
BE... BEEP!

ARE THERE
PIRANHAS IN
NEW JERSEY?

WOW!
THE METEOR
HIT 5 STARS!

I
WANNA
PROGRAM
A GAME
NOW...
CAN I?

DAD...



by multiplying two random numbers together. The while statement in shoot causes the motion. It repeatedly paints a new head, and erases the tail. How far back the erasing is done is determined by a variable called "big". For increasing values of k, the spot at row k and column start+delta*k is painted. This extends the shooting star down one step. Then the spot painted big steps ago is erased. The first big times this erasure is off screen, but that does no harm. When the erasure is off screen and k is larger than big, then the shooting star has completely traversed the screen. This causes a return, which leaves both the while, and shoot itself.

Notice that if a shooting star makes a direct hit on a fixed star, the fixed star is erased simultaneously with the tail of the shooting star.

For added interest, large shooting stars, called cruisers, erase not only direct hits, but also any fixed stars they merely approach. The last if statement does this with two plots.

FIGURE 4-4

```

/*
/* Meteor shower program, by T. A. Gibson, Nov. 1977.
/* Demonstrates use of random and plot.
star[
  clean
  pl"give a pattern number "
  seed=last=gn
  int k,stars
  int shoots
  ps" How many fixed stars "
  stars=gn
  ps" How many shooting stars "
  shoots=gn
  clean
  while((k=k+1)<stars)plot random(1,46),random(1,126),1
  k=0
  while((k=k+1)<shoots)shoot
]
shoot[
  int k
  int start
  start=random(20,107)
  int big
  big=random(0,2)*random(0,2)
  int delta
  delta=random(-3,+3)
  while(1)[
    k=k+1
    plot k,start + delta*k , 1
    if(plot (k-big, start + delta*(k-big), 0)) if(k>big) return
    if(big>3)[
      plot k-big, start+1+delta*(k-big),0
      plot k-big, start-1+delta*(k-big),0
    ]
  ]
]

```

End of FIGURE 4-4

V. INTERNAL OPERATION OF THE TINY-C INTERPRETER

For the student, the merely curious, or the software hack, this chapter describes the internal operation of the tiny-c interpreter. Software purchasers have a right to a thorough explanation of the operation of the product, INCLUDING commented source code.

Currently there are two implementations of tiny-c: the 8080 and the PDP-11. Both programs implement the same data structure and control flow. Both contain essentially the same set of subroutines.

In the narrative parts of this chapter, we describe the action of each of these subroutines. There are four parts to each description: arguments, results, errors, action.

ARGUMENTS describe data given to the subroutine through its calling sequence, e.g., via registers or the machine stack. We do not say in the narrative how arguments are passed, as that varies among different implementations. Refer to the listings for these details. Some routines will also use data in global cells such as CURSOR and PROGEND. These are not listed as arguments. Their use is described under the action part of the narrative.

RESULTS are data given back to the calling routine. This does not include modifications to the global cells. Many routines return no results to the calling routine. This does not mean they don't do anything! Don't confuse results (an explicit signal to the calling routine) with action (what the called routine does.)

ERRORS refers to errors detected by the interpreter in the tiny-c program being processed. All errors result in a call to ESET. Many routines detect no errors.

ACTION describes what the routine accomplishes.

The narrative has been kept general, so it will apply to all implementations, even improved ones. For example, a faster

search algorithm could be installed in ADDRVAL, but the narrative describing ADDRVAL would remain the same. The way we accomplish this is to say WHAT ADDRVAL does, NOT HOW it does it. The what-not-how approach to software description assumes the reader can read listings to determine the explicit algorithms used.

Section 5.12 is the sole exception. It describes subroutines needed only on eight-bit processors. Currently this discussion applies solely to the INTEL 8080. These routines are not in the PDP-11 version, nor are they likely to be necessary in any processor with full 16-bit arithmetic.

We begin this chapter with a discussion of tiny-c data objects.

5.1 Data Objects

There are two data declarations in tiny-c, int and char. They declare data objects which have five intrinsic properties: name, class, size of one datum, length, address where allocated.

5.1.1 Name

The name of an object is given in its declaration. For example, in

```
int X(10)
```

the object has name X. At most, 8 characters of a name are remembered, the first 7 and the last. Longer names may be used but their 8th through next-to-last character mean nothing. Thus

```
WHATHAPPENED  
WHATHAPD  
WHATHAPHAZARD
```

all appear to tiny-c as the same name. The second of these is the canonicalized eight-byte form.

5.1.2 Class

There is a difference between X and Y in:

```
char X, Y(0)
```

X is a character datum. Y is the address of a character datum.

The number of addresses one must go through to get an actual datum is the CLASS of the object. So X has class 0, and Y has class 1. You can think of class as the dimension of the object when viewed as a matrix. Thus X has dimension 0, and Y dimension 1.

Y(0) also has a class attribute. Y(0) is a character datum. So it has class 0. Consider

```
int VECTOR(80)
```

VECTOR has class 1. VECTOR(17) has class 0.

Classes greater than 1 theoretically exist but are not implemented in tiny-c.

5.1.3 Size

Size refers to the size of the ultimately referenced object. It does not refer to the size of an array. All character data has size 1. All integer data has size 2. Thus in

```
char LETTER, LINE (80)
```

both LETTER and LINE have size 1.

5.1.4 Length

The length of an object is measured in number of data items, not bytes. All class 0 objects have length 1. Class 1 objects have lengths equal to the declared size of the array.

Note that this is one larger than the number written, because element 0 must be counted. Thus,

```
char LINE (80)
int VECTOR (80)
```

both have length 81.

5.1.5 Address

The address of an object is the lowest memory byte address where the object's value(s) is(are) stored. An n byte (size * length = n) object has its value(s) stored in address, address + 1, ..., address + n-1.

5.2 The Variable Table

All functions and all currently active variables are described in the variable table. An entry in the variable table is 14 bytes:

BYTES	ENTRY
0-7	eight-byte name (canonicalized)
8	class
9	size (of one datum)
10-11	length (in data elements)
12-13	address of first value

A class value of ASCII 'E' for entry represents a function name. Size and length are ignored. The address value is where the cursor must be placed to begin execution of the function.

The variable table is allocated from the address stored in BVAR to that stored in EVAR inclusive. BVAR and EVAR are in the installation vector.

Currently active variables are:

- all library symbols, including library functions
- all global symbols, including functions
- all local variables for currently active functions, including function arguments.

A function is active if it has been invoked, but has not completed. In the case of recursion, all incarnations count as active. All function names are either global or library, and all are entered in the variable table. Note that the function's name is entered without regard for its active or inactive status. The active function simply determines which local variables are currently active.

Suppose the following code is being interpreted, and control is currently in aaa:

```

char globalsymbol
main [
  int localsymbol
    aaa 2,3
    bbb
  ]
  aaa int x,y [
    int n
    .
    .      <----- current control point
    .
  ]
  bbb [
  char othersymbol
  .
  .
  .
  ]

```

The global symbols are:

```

globalsymbol
main
aaa
bbb

```


The active functions are main and aaa. Their local variables are:

```
localsymbol
x
y
n
```

These eight variables are the ones allocated in the variable table. Later when aaa returns, x, y and n are de-allocated. When bbb is invoked, anothersymbol is allocated. At that point six symbols are in the table:

```
globalsymbol
main
aaa
bbb
localsymbol
anothersymbol
```

5.3 Layers in the Variable Table

The variable table is organized into layers so only those symbols accessible to a function will be searched. A symbol is accessible if it is

- * local to the function, or
- * global, or in the
- * library.

This excludes variables local to active functions not currently executing. Thus:


```

int g
main [
    int x
    aaa
]
aaa [
    int y
    bbb
]
bbb [
    int z
    .
    . <----- current control point
    .
]

```

The variables `g` and `z` are accessible to `bbb`. `x` and `y`, although active, are not accessible to `bbb`.

Layer 0 is reserved for library symbols. Globals are in layer 1 initially, but can occur in other layers (See Section 5.10.11). After layer 1, if there are n currently active functions, i.e. a main program, a function invoked by main, a function invoked by that function, etc., n levels deep, then those n functions are in layers 2, 3, ..., $n+1$. The locals for main are in layer 2. The locals for the currently executing function are in layer $n+1$. This layer is called CURFUN.

When a symbol is needed, it is searched as follows:

```

First, all symbols in the CURFUN layer are searched.
    If not found there, then
Second, all symbols in the global layer are searched.
    If not found there, then
Last, all symbols in the library layer are searched.
    If not found there, a symbol error is raised.

```

Notice this order of search guarantees the properties in Section 2.3.2.

5.4 The Function Table

The mechanism for layering is the function table, allocated from the address stored in BFUN to that stored in EFUN. BFUN and EFUN are in the installation vector. An entry in the function table has three addresses (six bytes).

BYTES

- | | |
|-----|--|
| 0-1 | The address of the first variable table entry for this function. |
| 2-3 | The address of the last variable table entry for this function. |
| 4-5 | Backup pointer, explained below. |

The address in CURFUN points to the first byte of the function table entry of the currently executing function. The address in CURGLBL points to the first byte of the function table entry for current globals.

Library symbols are always in layer 0, and BFUN points to the first byte of that layer. So the implementation of the search order described above is simply to search symbols spanned by CURFUN, then CURGLBL, then BFUN.

5.5 Symbol Table Tools

The following subroutines manipulate and search the variable and function tables:

NEWFUN
FUNDONE
NEWVAR
ADDRVAL

These subroutines use the auxiliary subroutine

CANON

to canonicalize variable names to eight bytes.

5.5.1 NEWFUN

Arguments: None
Results: None
Errors: Too many functions
Action: Allocates a new layer in the function table,
with zero entries in the variable table.

5.5.2 FUNDONE

Arguments: None
Results: None
Errors: None
Action: Deallocates a layer in the function table.
Deallocates all variables in that layer.
Deallocates all value space seized within that
layer.

5.5.3 NEWVAR

Arguments: Canonicalized variable name, class, size, length,
and value.
Results: Address of first byte of allocated space.
Errors: Too many variables, or too many values.
Action: Enters a new variable in the current layer of
the variable table.

There are several considerations in allocating a value for a variable. If the variable is local or global, space is allocated and set to 0. If the variable is an argument to a function, and is class 0 (i.e. a single datum, not an array name) then space is allocated, and the passed value copied into the space. If the variable is an argument to a function, and is class greater than zero, then it is the address of an existing array. Space is not reallocated for these variables. The passed value - the array address - is entered as the new variable's address-of-first-value (bytes 12-13 of variable table). Thus, in effect, the array is passed.

Values are allocated in program space. PROGEND points to the last byte used for program. PRUSED points to the last byte

used for program and values. EPR points to the last byte of program space. EPR is in the installation vector.

5.5.4 ADDRVAL

Arguments: Canonicalized variable name.
 Results: Address of first byte of allocated space, class, size, and length.
 Errors: Variable name not found.
 Action: Searches for the variable name and returns its properties. Searches locals, globals, and library symbols in that order.

5.5.5 CANON

Arguments: Addresses of first and last bytes of an uncanonicalized variable name and the address of an eight-byte buffer.
 Results: Canonicalized variable name in buffer.
 Errors: None
 Action: Eight bytes are placed in the buffer. If the character string is shorter than eight bytes, it is padded on the right with nulls. If the variable name is longer than eight bytes, the first seven bytes and the last byte are placed in the buffer.

5.6 The Tiny-c Stack

Tiny-c has a stack for use in evaluating expressions. This stack is implemented in software, and is not to be confused with the 8080 or PDP-11 machine stack. This stack, called the tiny-c stack, is allocated from the address stored in BSTACK to the address stored in ESTACK. BSTACK and ESTACK are in the installation vector.

An entry on the tiny-c stack is 5 bytes:

BYTE	ENTRY
0	class, defined in Section 5.1.2
1	lvalue, defined below
2	size, defined in Section 5.1.3
4-5	stuff, defined below

An lvalue (left-side-value) is any symbol that can be written to the left of an equal sign. Allowable lvalues include variable names, subscripted array names, and unsubscripted array names.

The variable names and subscripted array names have class 0, and their value is a single character or integer datum. The unsubscripted array names have class 1, and their values are addresses.

Processing of an expression is done left to right. Whenever an lvalue is recognized, its attributes are put on the tiny-c stack. Bytes 4 and 5, stuff, are set to the address where the value is stored. An ASCII 'L' is put in byte 1, lvalue, signifying stuff is a REFERENCE to the actual value.

Whenever a constant is processed, or the results of arithmetic must be stacked, then the actual data value is put into stuff. An ASCII 'A' is put into lvalue, signifying stuff IS actual data.

Consider the processing of the following expression

$$K = J + 1$$

where K and J are both integers. Suppose K has address 4000, J has address 4002, and J has value 7.

STEP	ELEMENT		CLASS	TINY-C STACK		STUFF
				LVALUE	SIZE	
1	K	Top --->	0	L	2	4000
2	J	Top --->	0	L	2	4002
			0	L	2	4000
3	1	Top --->	0	A	2	1
			0	L	2	4002
			0	L	2	4000

At this point the plus operation is performed. First, the 1 is popped off the stack. It is an actual value, so it is left as a 1. Then the 4002 is popped. It is an lvalue so the contents of addresses 4002 and 4003 (two bytes because size is 2) are retrieved. This is the value of J, 7. They are added, and the sum pushed as an actual:

STEP		CLASS	TINY-C LVALUE	STACK SIZE	STUFF
4	Top --->	0	A	2	8
		0	L	2	4000

Now the equal operation is performed. The top entry, 8, is popped and held. The next entry is popped. It must be an lvalue but it is not converted to an actual, because the current value of K is irrelevant. Only the address of K is needed, and that is 4000. The held 8 is stored in bytes 4000 and 4001, thus performing the assignment.

The value of the expression is 8. (Recall that all expressions, including those with an equal sign, have values.) So the last step in processing this expression is to push the 8 back on the stack as an actual.

STEP		CLASS	TINY-C LVALUE	STACK SIZE	STUFF
5	Top --->	0	A	2	8

5.7 Stack Tools

The following subroutines manipulate the tiny-c stack:

```

PUSH
PUSHK
PZERO
PONE
POP
TOPTOI
POPTWO
TOPDIF
EQ

```

5.7.1 PUSH

Arguments: Class, lvalue, size, stuff
 Results: None
 Errors: Stack full
 Action: The arguments are pushed onto the stack as its top entry.

5.7.2 PUSHK (8080)

Arguments: Stuff
Results: None
Errors: Stack full
Action: Stuff is pushed onto the stack with class 0, lvalue A, size 2.

5.7.3 PZERO, PONE (8080)

Arguments: None
Results: None
Errors: Stack full
Action: Same as PUSHK with stuff 0 for PZERO and 1 for PONE.

5.7.4 POP (8080)

Arguments: None
Results: Class, lvalue, size, stuff
Errors: None
Action: The stack is popped and the results returned.

5.7.5 POP (PDP-11)

Arguments: None
Results: Stuff
Errors: None
Action: The stack is popped and stuff returned.

5.7.6 TOPTOI

Arguments: None
Results: An actual value in integer form.
Errors: None
Action: The stack is popped. If the popped entry is an lvalue, it is converted to an actual. Otherwise, the datum in stuff is used. If the resulting datum is of size 2 or of class greater than 0 (i.e., a two-byte address) it is returned. If the datum is of size 1 and class 0, it is first converted to an equivalent two-byte integer and then returned.

5.7.7 POPTWO (8080)

Arguments: None
Results: Two integers
Errors: None
Action: TOPTOI is called twice and both of its results are returned.

5.7.8 TOPDIF (8080)

Arguments: None
Results: An integer
Errors: None
Action: The top two layers are popped and their difference (next-to-top minus top) returned.

5.7.9 EQ

Arguments: None
Results: None
Errors: Left-value error
Action: The top level is popped and converted to an actual if necessary. The next level is popped and must be an lvalue, or an lvalue error is raised. The value referenced by the lvalue is replaced by the actual. Eight bits are replaced if the lvalue represents a class 0 eight-bit datum (i.e. a character). 16 bits are replaced if the lvalue represents a class greater than 0 (i.e. an address) or a class 0 16-bit datum (i.e. an integer).

5.8 Scanning Tools

The scanning tools are subroutines used by the tiny-c interpreter to recognize tokens of the language. They do not themselves define the language, and in fact could be used in parsers of languages other than tiny-c.

Most of these tools make use of the contents of the global variable CURSOR. This is the address of the next character to be examined in the tiny-c program being interpreted. We

say that "cursor points to" that character. Some of these tools "advance the cursor", i.e., increment the address in CURSOR, so that it points to a character further on in the program.

5.8.1 LIT

Arguments: The address of the first byte of a character string which is to be matched. The string is terminated by a null byte.

Results: A flag signalling match or no match.

Errors: None

Action: The cursor is first advanced over blanks. If the next characters match the literal, then the cursor is advanced beyond them and a match is signalled. Otherwise, the cursor remains on the first non-blank character and no match is signalled.

5.8.2 SKIP

Arguments: A left delimiter and a right delimiter, each a single character.

Results: None

Errors: Cursor runaway

Action: The cursor is advanced beyond a balanced pair of left and right delimiters. It is assumed that one left delimiter has already been matched. Cursor runaway occurs when a character beyond PROGEND is being examined.

5.8.3 SYMNAME

Arguments: None

Results: A flag signalling match or no match.

Errors: None

Action: The cursor is first advanced over blanks. Then, if the next set of characters is a symbol, the cursor is advanced beyond it, the global variable FNAME is pointed to its first character, the global variable LNAME to its last, and a match is signalled. Otherwise, the cursor remains on the first non-blank character

and a no match is signalled. A symbol is an alphabetic character followed by an arbitrary number of alphanumeric characters.

5.8.4 CONST

Arguments: None
Results: A flag signalling match or no match and the type of constant matched.
Errors: Cursor runaway
Action: The cursor is first advanced over blanks. Then one of three types of constants is matched:

1. Number: an optional + or - followed by any number of digits.
2. Quoted String: "anything" or "anything null-byte"
3. Primed String: 'anything'

If a number is matched, FNAME and LNAME are pointed to its first and last significant characters, including sign if any. The cursor is advanced beyond the last digit.

If a quoted string is matched, FNAME is pointed to the first byte AFTER the first quote. LNAME is not used. The second quote is replaced by a null byte. The cursor is advanced beyond this null. This makes string constants conform to the tiny-c convention that character strings end with a null.

If a primed string is matched, FNAME points to the first byte AFTER the first prime. LNAME is not used. The second prime is not modified. The cursor is advanced beyond the second prime.

5.8.5 REM

Arguments: None
Results: None
Errors: Cursor runaway
Action: Advances cursor past all remarks, carriage returns, and blanks.

5.8.6 BLANKS

Arguments: None
Results: None
Errors: None
Action: Advances cursor beyond blanks.

5.9 The Statement Analyzer

The tiny-c statement analyzer is a collection of eleven subroutines which analyze and evaluate tiny-c statements. Figure 5-1 shows the calling relationship among the ten routines.

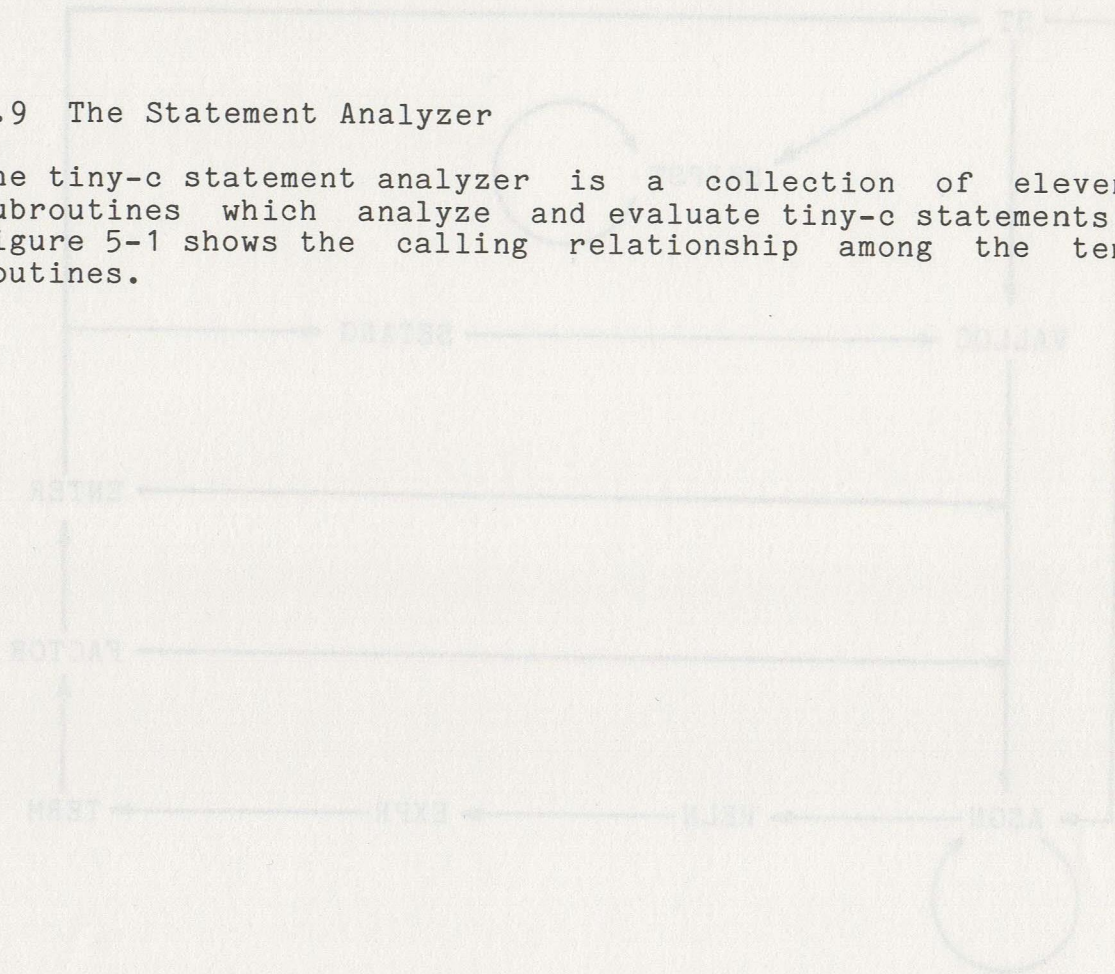
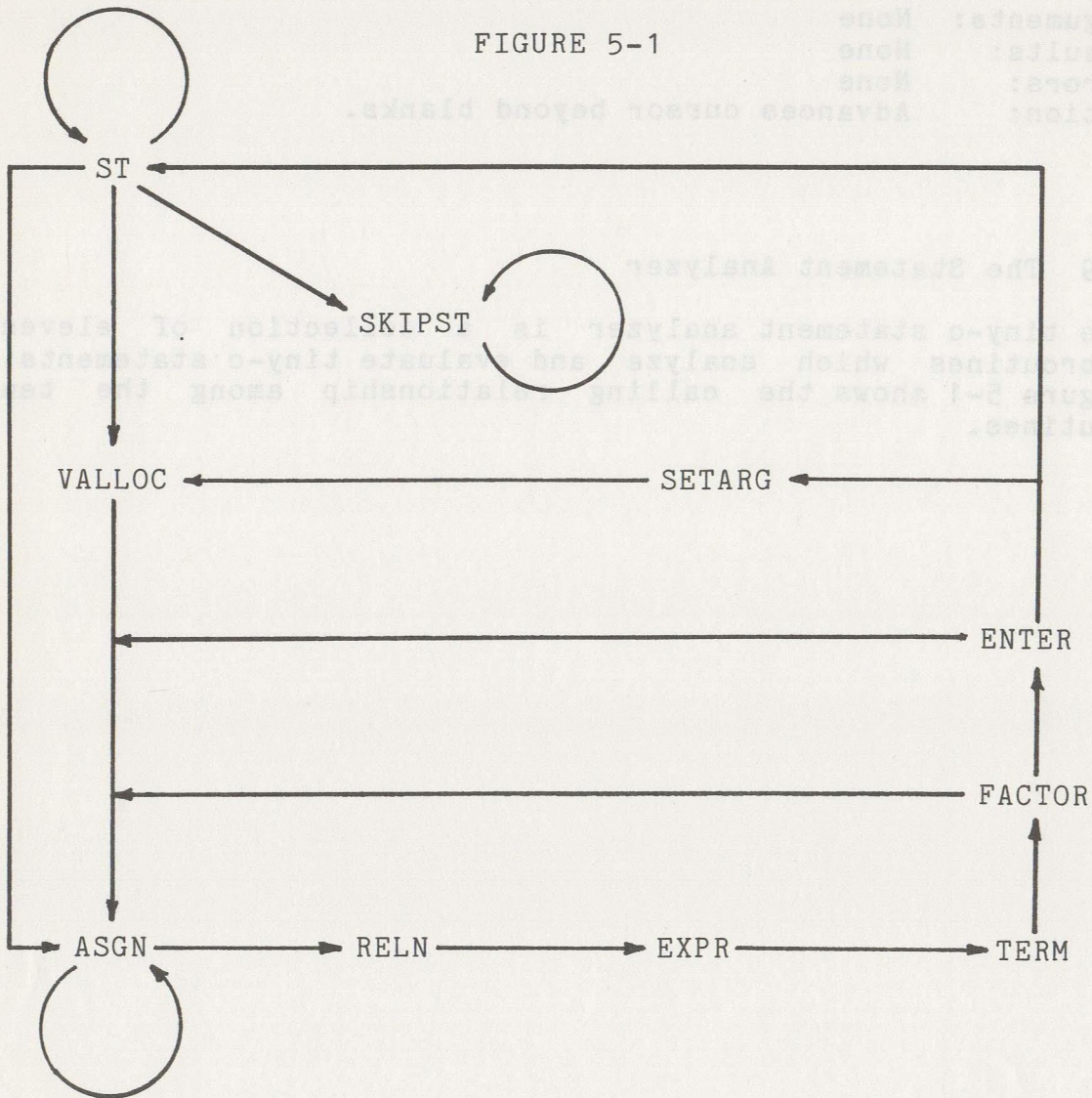


FIGURE 5-1



End of FIGURE 5-1

Briefly, the action of each of the routines is as follows:

- ST -- determines the kind of statement currently being analyzed and keeps track of the level of statement compounding.
- SKIPST -- moves the cursor beyond a complete, possibly compound, statement without interpreting it.
- VALLOC -- parses variables in int and char statements, and calls NEWVAR to allocate them.
- ASGN -- matches expressions and calls EQ to perform any assignments.
- RELN -- evaluates an expression by calling EXPR. If a relational operator follows the expression, evaluates the next expression and then evaluates the relation between the two expressions.
- EXPR -- evaluates unary plus and minus of a term and binary plus and minus between terms. Calls TERM to evaluate terms.
- TERM -- evaluates multiplication, division, and remainder operators between factors. Calls FACTOR to evaluate factors.
- FACTOR -- evaluates a factor which is an expression enclosed in parentheses, a constant, or a variable name including a function call. Evaluates function names by calling ENTER.
- ENTER -- transfers control to a function by stacking the current position of the cursor and setting the cursor to the beginning of the function. Allocates a new layer (Sections 5.3 and 5.4) in the function table by calling NEWFUN and assigns argument values to local variables by calling SETARG.
- SETARG -- assigns function argument values in a function call to the local variables used in the definition of the function. Enters local variable names in the variable table by calling VALLOC.
- LINK -- enters global names in a new layer of the variable table.

Another view of the action of each statement analyzer routine is provided by examining the characters and character strings that each routine recognizes and past which it moves the cursor. Significant cursor movement routines, i.e., routines which move the cursor beyond things other than blanks and remarks, are LIT, CONST, SKIPST, and SYMNAME. Of these, only LIT can move the cursor beyond different strings of symbols depending on its argument. In Figure 5-2 we display the characters with which each statement analyzer routine calls LIT and if any of these routines calls CONST, SKIPST, or SYMNAME, we indicate this fact by enclosing the routine name in angle brackets, < >.

FIGURE 5-2

ST	int	char	if	else	while	return
	break	,	;	[]	()	<skipst>
ENTER	int	char	()	,	;	
VALLOC	()					
ASGN	=					
RELN	<=	>=	==	!=	>	<
EXPR	+	-				
TERM	*	/	%			
FACTOR	()	MC	<const>	<symname>		
SKIPST	if	else	while	[	(	;

End of FIGURE 5-2

The statement analyzer routines communicate with each other by using the tiny-c stack almost exclusively. What is done with the contents of the stack is determined by the position of the cursor and, in the case of FACTOR and VALLOC, the setting of the global variables FNAME and LNAME. In the following descriptions, the term "matches" means the routine "recognizes and advances the cursor over the text of", while the term "evaluates" means the routine actually "computes the value of".

5.9.1 ST

Arguments: None
Results: None
Errors: Program segment encountered which is not a tiny-c statement.
Action: Matches a tiny-c statement. Calls VALLOC to enter global variables in variable table, ASGN to evaluate relational expressions and ST to evaluate components of compound statements. Uses SKIPST to skip statements which do not need to be evaluated.

5.9.2 VALLOC

Arguments: A class value and a value of that class.
Results: Address of first byte of allocated space.
Errors: Invalid symbol name or invalid array size defining expression.
Action: Matches a variable declaration and enters its name and attributes in the variable table by calling NEWVAR. The initial value of the variable name is the passed value of that class. In the case of arrays, evaluates the expression defining the size of the array.

5.9.3 ASGN

Arguments: None
Results: Success flag
Errors: None
Action: Matches an expression. In the case of an assignment operator calls EQ to make the assignment.

5.9.4 RELN

Arguments: None
Results: None
Errors: None
Action: Matches an expression and, if it is followed by a relational operator, matches another expression and evaluates the relational operator.

5.9.5 EXPR

Arguments: None
Results: None
Errors: None
Action: Matches an optional unary plus or minus followed by one or more terms separated by binary pluses or minuses. If any operators are matched, the expression is evaluated.

5.9.6 TERM

Arguments: None
Results: None
Errors: None
Action: Matches one or more factors separated by multiplication, division, or remainder operators. If any operators are matched, the term is evaluated.

5.9.7 FACTOR

Arguments: None
Results: None
Errors: Expressions without balanced parentheses; class 0 variables with subscripts; undeclared variable names, or unrecognized statement syntaxes.
Action: Matches constants, variable names including function calls, and expressions enclosed in parentheses. Evaluates constants and variables, and uses ENTER to evaluate function calls.

5.9.8 ENTER

Arguments: An address
Results: None
Errors: Number of arguments in function call not equal to number of arguments in function definition.
Action: Saves the cursor, allocates a layer in the function table by calling NEWFUN, calls SETARG to assign function argument values to function local variables, sets cursor to the passed address, matches parameter declarations and assigns them space and values using SETARG, pops all arguments off the stack, and calls ST. Upon return from ST, restores the cursor, and deallocates the layer from function table by calling FUNDONE.

5.9.9 SETARG

Arguments: The address of an entry in the tiny-c stack and a class value.
Results: None
Errors: None
Action: Converts the tiny-c stack entry to an actual and calls VALLOC with this value and the passed class value to enter an argument in the variable table as a local variable with an initial value equal to the value passed in the tiny-c stack.

5.9.10 SKIPST

Arguments: None
Results: None
Errors: Cursor runaway
Action: Advances the cursor to the end of a tiny-c statement and beyond any remarks which appear at the end of the statement. Leading remarks are also skipped.

5.9.11 LINK

Arguments: None
Results: None
Errors: Unidentifiable global declaration or function name not succeeded by a left bracket.
Action: Allocates a new layer of the variable table. Enters global variable names, including all function names, in the variable table. If an "endlibrary" statement is encountered, a second new layer is allocated and the entering of global variable names continues.

5.10 Standard Machine Calls

These Machine Calls are furnished with the tiny-c interpreter. They are always loaded and available for use. They are called as described in Section 2.10. The function number is the LAST argument. Thus MC 1 is called like this:

MC 'x', 1

Where no results are indicated, a 0 is returned as the value of the MC.

5.10.1 MC 1 ... PUTCHAR

Arguments: A character value
Results: None
Errors: None
Action: The character is transmitted to the console terminal.

5.10.2 MC 2 ... GETCHAR

Arguments: None
Results: A character value
Errors: None
Action: A character is retrieved from the console terminal and returned as the value of the function. The input port from the console terminal is cleared to receive another character.

5.10.3 MC 3 ... FOPEN

Arguments: A mode, filename, file size, and channel number.
Results: An open status indicator
Errors: None
Action: Same as fopen in the standard library, see Section 2.9.1.

5.10.4 MC 4 ... FREAD

Arguments: Pointer and a channel number
Results: A get status indicator
Errors: None
Action: Same as fread in the standard library, see Section 2.9.1.

5.10.5 MC 5 ... FWRITE

Arguments: Two pointers and a channel number
Results: A put status indicator
Errors: None
Action: Same as fwrite in the standard library, see Section 2.9.1.

5.10.6 MC 6 ... FCLOSE

Arguments: A channel number
Results: None
Errors: None
Action: Same as fclose in the standard library, see Section 2.9.1.

5.10.7 MC 7 ... MOVEBL

Arguments: Two pointers and an integer
Results: None
Errors: None
Action: Same as movebl in the standard library, see Section 2.9.1.

5.10.8 MC 8 ... COUNTCH

Arguments: Two pointers and a character value
Results: The number of appearances of the character between the two pointers inclusively.
Errors: None
Action: Same as countch in the standard library, see Section 2.9.1.

5.10.9 MC 9 ... SCANN

Arguments: Two pointers, a character value, and an integer
Results: A pointer to the last character examined
Errors: None
Action: Same as scann in the standard library, see Section 2.9.1.

5.10.10 MC 10 ... INTERRUPT

Arguments: None
Results: None
Errors: None
Action: A processor stop is executed, or a return to the processor's operating system.

5.10.11 MC 11 ... ENTER

Arguments: Four pointers, called FACTS, START, FIRST, and LAST in that order.
Results: None
Errors: None
Action: An application program is started. The text of the program is from FIRST to LAST inclusive. LAST should point to a statement terminator; a carriage return is recommended. START points to

a character string, which is a tiny-c statement. The statement can be within the program text, but need not be. FACTS points to a three-byte data area where facts are returned regarding why the application program stopped. The first two bytes of the area are the error code, the next two, the value of CURSOR when the program stopped.

The program text is linked (See Section 5.9.11 LINK). This causes a new layer of variables to be allocated for the variables defined by the link process. This layer is defined as the new Current Global area (See Sections 5.2, 5.3, 5.4). Another layer is allocated for the first layer of locals needed by the application.

The application level (global variable APPLVL) is incremented (See Section 4.3.3).

Then, if no errors occurred in these preliminary actions, the statement at START is executed. It has access to the newly defined globals. It follows that, if the statement is a function call to a program defined in the application text, that program will be called.

Upon completion, all global variables affected by MC 11 are restored to their previous values. If an error occurred in the application, that fact is captured in the bytes referenced by FACTS. The global variable ERR is set back to 0, reflecting the fact that no error has occurred in the program that called MC 11. The Current Global area and the Current Function layer are restored to their previous values. Several other pointers used in the interpretation process are restored, and the invoking function is resumed.

Values from the application are allocated from LAST+1 up.

5.10.12 MC 12 ... CHRDY (8080)

Arguments: None
Results: A character value
Errors: None
Action: Same as chrdy in the standard library, see
Section 2.9.1.

5.10.13 MC 13 ... PFT

Arguments: Two pointers
Results: None
Errors: None
Action: Same as pft in the standard library, see
Section 2.9.1.

5.10.14 MC 14 ... PN

Arguments: An integer
Results: None
Errors: None
Action: Same as pn in the standard library, see
Section 2.9.1.

5.11 Other Tools

5.11.1 ALPHA

Arguments: An address
Results: A flag signalling valid or invalid alphabetic
character.
Errors: None
Action: If the content of the address is a valid alpha-
betic character, the valid flag is returned.
Otherwise the invalid flag is returned.

5.11.2 ALPHANUM

Arguments: An address
Results: A flag signalling valid or invalid alphanumeric character.
Errors: None
Action: If the content of the address is a valid alphanumeric character, the valid flag is returned. Otherwise the invalid flag is returned.

5.11.3 ATOI

Arguments: An address
Results: An integer value
Errors: None
Action: Starting at the argument, converts the string of numeric characters in sequentially higher addresses to the corresponding integer value. The string of numeric characters may be preceded by leading blanks and immediately preceded by a minus sign. Evaluation is terminated by the first non-numeric character. If no numeric characters are encountered, a value of 0 is returned.

5.11.4 BZAP (8080)

Arguments: Two memory addresses, the second larger than the first, and a one-byte value.
Results: None
Errors: None
Action: Like ZERO (see Section 5.11.14), except the one-byte value is put into memory instead of zeroes.

5.11.5 CEQ

Arguments: Two addresses
Results: A flag signalling match or no match
Errors: None
Action: Tests if the character string at the first address is a leading substring of the character string at the second address. Signals a match if it is, otherwise no match. The ends of the strings are signalled by a null byte.

5.11.6 CEQN

Arguments: Two addresses and an integer, n
Results: A flag signalling match or no-match
Errors: None
Action: Tests if the contents of the n sequentially higher addresses starting at the first address, are equal to the contents of the n sequentially higher addresses starting at the second address. Signals a match if all contents are equal.

5.11.7 CTOI (PDP-11)

Arguments: An address
Results: An integer value
Errors: None
Action: Returns contents of the address as an integer value.

5.11.8 ESET

Arguments: Error number
Results: None
Errors: None
Action: If and only if ERR is 0, the global variable ERR is set to the error number, and the global ERRAT to the value of CURSOR. Otherwise, no action is taken.

5.11.9 MOVN (PDP-11)

Arguments: Two addresses and an integer, n
Results: None
Errors: None
Action: Copies the contents of n sequentially higher addresses starting at the first address to the n sequentially higher addresses starting at the second address.

5.11.10 NUM

Arguments: An address
Results: An integer value
Errors: None
Action: If the content of the address is a numeric character, its integer value is returned. Otherwise, -1 is returned.

5.11.11 RETURN (PDP-11)

Arguments: None
Results: None
Errors: None
Action: Restores registers 2, 3, 4, and 5 and branches to the address that was in 2(R5) before R5 was restored. When used in conjunction with SAVE, and JMPed to in place of an RTS subroutine return, subroutine calling can be recursive.

5.11.12 SAVE (PDP-11)

Arguments: None
Results: None
Errors: None
Action: Saves registers 2, 3, 4, and 5 on the processor stack. Leaves R5 pointing to the saved value of R5. Arguments passed on the processor stack can be accessed as 4(R5), 6(R5), etc. SP, R2, R3, and R4 can be used freely. When used in conjunction with RETURN, subroutine calling can be recursive.

5.11.13 TCTOI (PDP-11)

Arguments: Two addresses
Results: An integer value
Errors: None
Action: Returns the contents of two bytes of memory as an integer.

5.11.14 ZERO (8080)

Arguments: Two memory addresses, the second larger than the first.
 Results: None
 Errors: None
 Action: Memory is cleared from the first address to the second, inclusive.

5.12 16-Bit Arithmetic Tools (8080)

5.12.1 DNEG

BC <--- - (BC).

(Read this as BC is replaced by minus the contents of BC.) Uses A.

5.12.2 DENEG

DE <--- -(DE).

Uses A.

5.12.3 HLNEG

HL <--- - (HL).

Uses A.

5.12.4 BCRS

BC <--- (BC) >> 1.

(Read this as BC is replaced by the contents of BC right-shifted one bit.) High bit of B is replaced by a 0. Z is set if and only if result is 0. Uses A.

5.12.5 DELS

DE <--- (DE) << 1.

E's low bit <--- 0. Z is set if and only if the result is 0. Uses A.

5.12.6 RDEL

DE <--- (DE) << 1.

E's low bit <--- CY. Z is set if and only if the result is 0. Uses A.

5.12.7 HLLS

HL <--- (HL) << 1.

Low bit of L <--- 0.

5.12.8 DADD

DE <--- (DE) + (BC).

Flags set as in DSUB. Uses A.

5.12.9 DSUB

DE <--- (DE) - (BC).

Z is set if and only if the result is 0. CY is set if and only if the result is negative (high order bit of D becomes 1). Uses A.

5.12.10 DMPY

DE <--- (DE) * (BC).

Uses all registers.

5.12.11 DDIV

DE <--- (DE)/(BC).
HL <--- (DE) % (BC).

Uses all registers.

5.12.12 DREM

DE <--- (DE) % (BC).
HL <--- (DE)/(BC).

Uses all registers.

5.12.13 DCMF

The flags Z, S, and CY are set by usual 8080 conventions depending on the result of (DE) - (BC).
Uses A. No other registers are changed.

VI. INSTALLING TINY-C ON YOUR 8080 SYSTEM

See end of this section for POLY-88 version installation.

If you have an installed load-and-go version of tiny-c, you can skip this chapter. If you have an uninstalled version, then you must follow the installation procedure given here. It is a technical job. You must have an understanding of machine language, terminal input/output, and file input/output techniques. You must know the details of your operating system's terminal and file input/output subroutines (if they exist) or be ready to design and implement them if they do not. If you have this know-how already, the installation should take two or three days. Otherwise, the job may take some weeks.

The uninstalled tiny-c lacks seven subroutines for doing terminal and file input/output. The specifications for these routines are given in Section 6.1, but they must be coded by the installer. An installation vector in tiny-c contains jumps to these seven subroutines along with other installation dependent data. The addresses of the seven subroutines must be placed in these jump instructions.

Altogether, there are nine steps to installing a complete tiny-c on your 8080. They are:

1. Implement and test the four file interface routines and three terminal interface routines.
2. Load and relocate tiny-c.
3. Load the interface routines.
4. Link tiny-c to the interface routines using the installation vector.
5. Modify the other installation vector elements if necessary.

6. Write the results to your mass storage.

7. Test the results so far.

You now have a raw tiny-c which can execute programs, but not prepare them or modify them. To develop tiny-c programs, you may use either your own editor or the Program Preparation System (PPS) provided with tiny-c. To use the PPS,

6'. Load the results of step 6, if they're not already loaded.

8. Load tc.pps into the tiny-c program space. This file contains the PPS, a tiny-c program to prepare, test, and edit other tiny-c programs.

9. Write the results (including tc.pps) to mass storage.

You now have a tiny-c Program Preparation System.

The rest of this chapter expands on the nine steps. Section 6.10 gives suggestions on what to do if things go wrong. We urge you to carefully check each step of the installation before proceeding to the next step.

6.1 STEP 1 -- The Interface Routines

There are seven i/o interface routines: three to the terminal, and four to the file system. These must be implemented by the installer. In the happiest case, some or all exist already in your operating system. In an extreme case, you may have to code them yourself. Most likely, your operating system has subroutines that perform all the needed functions. All you will need to do is provide a little bit of code to adapt the seven interfaces to these operating system routines. An example of this is given later in Section 6.1.2.

Each routine is CALLED by tiny-c. Each must return control when finished by using the RET instruction.

6.1.1 Specification of the 8080 Terminal Interface

INCH

No arguments. This subroutine pauses until a character is available from the keyboard. It returns the ASCII character in register A.

CHRDY

Tests whether or not a character is ready from the terminal. It does NOT WAIT for a character. If no character is ready, a 0 is returned in A, and the Z flag is set. If a non-null character is ready, a copy of the character is returned in A, and the Z flag is not set. If a null character is ready, then a 1 is returned in A, and the Z flag is not set. Note that the character is not "read". To read the character (i.e., to remove it from the buffer, clearing the way for another character) INCH must be called. The fact that the ready flag happens to be a copy of the character should not be confused with the act of reading the character.

OUTCH

The A register contains an ASCII character. It is transmitted to the terminal. Nothing is returned. A, DE, and HL must not be modified.

6.1.2 Specification of the 8080 File Interface

FOPEN	HL	points to a character string of any length terminated by a null byte. This is the file name.
	DE	is an integer, specifying the file size in bytes. This is used to allocate space for newly created files.
	BC	is an integer, a unit (or device, or channel) number.
	A	1 for read, 2 for write access.

Prepare to read or write a file. If your OS needs to know whether this open is for reading or for writing, this infor-

mation is signalled in A. Otherwise, A can be ignored. If your OS supports multiple units, BC is a unit number. Otherwise it can be ignored. You may assume BC is a small integer (1, 2, 3, but not 197, 2727, 4793). Thus it can be used to address a file table if your design needs one. You may assume the unit designated in BC is not in use, i.e., if it was once opened it was subsequently closed before this call to FOPEN. If your OS allocates space for a file, and if $A == 2$, then DE can be used for this allocation. Otherwise, DE can be ignored. In any case DE is garbage if $A == 1$. If your OS supports file names, HL points to the beginning of a character string with the file name. Otherwise HL can be ignored. A result code must be returned in A and in the Z flag. 0 means FOPEN was successful. Nonzero means a problem has occurred. If a problem occurs, FOPEN must advise the user of the problem, because tiny-c will not.

FREAD	HL	points to first byte of the block to be read.
	BC	is a unit number

One record is read from the file opened on unit BC. The record may be any size from 1 to 256 bytes, according to what was written. A result code must be returned in register A and in the Z flag. On a successful read A must be set to 0, and Z set, and the number of bytes read is returned in DE. HL is left unchanged. A result code of -1 in A, and Z reset, means an end-of-file was read, or an attempt was made to read beyond the file's end. This result is normal and must not cause fatal consequences, e.g., aborting tiny-c. If Z is reset and A is anything except 0 or -1, it means an error has occurred.

FWRITE	HL	points to first byte.
	DE	points to last byte.
	BC	is a unit number.

The bytes from (HL) to (DE) inclusive are to be written as a record on the file opened on unit (BC). You may assume $(HL) \leq (DE)$ and the length $(HL) - (DE) + 1$, is less than or equal to 256. You must write the record so that the length can be determined by FREAD. If units are not supported, BC can be ignored. The result code returned in register A is 0 for a successful write and 1 otherwise.

FCLOSE BC is a unit number.

The file (connected to unit BC, if units are supported) is closed. You may assume no further reads or writes to this file or this unit will occur, unless named in a subsequent FOPEN. On a tape system, an end-of-file record is written.

6.1.3 File System Implementation Options

The file system has been specified to be as universally implementable as possible. Enough information and opportunity for control have been given to link to a sophisticated file manager. But the interface is uncomplicated enough to utilize even a very simple cassette recording system.

If you already have a tape operating system (TOS) or a disk operating system (DOS), you may have to implement only a small amount of linkage code. The operating system (OS) will do all the work.

FOPEN: If your OS has a file opening ability, you can connect with it here. If it needs to know whether the open is for read or write, the information is available. The file name is available if your OS uses file names. You may have to write code to modify the format and register assignments of this data before invoking your OS's file open. Units are for OSs supporting more than one open file.

If your OS doesn't have a file open, then FOPEN just returns 0. If reads and writes require file names, then FOPEN can simply capture the file name. An example of this is shown below.

FREAD: For cassette storage, FREAD should start the motor, read one record, then stop the motor. It must determine the number of bytes in the record, and return this number in DE.

FWRITE: Must record the record's length, probably in a record header (an "invisible" byte at the record's beginning). Some OSs do this already. FWRITE should also stop and start the motor for a cassette system.

FCLOSE: For a TOS, FCLOSE is where you write an end-of-file. Note that FCLOSE will be called at the end of a read-only access (A == 1 in FOPEN). You may need to make end-of-file

writing conditional on having a write access ($A == 2$ in FOPEN). On some file systems, such as a cassette system, an unconditional writing of an end-of-file in FCLOSE is satisfactory. For a DOS, FCLOSE is where you flush buffers and update the directory. Most DOSs have a close file routine to do this.

The tiny-c Program Preparation System never uses more than one file at a time and it always uses unit 1. Many applications will never use more than one file at a time. If your OS can only deal with one file, then ignore the unit number in BC; but, if you can handle multiple files, this number is the "key" that ties opens to reads, writes, and closes. You can assume it will be a small positive number. Thus you can use it, for example, to index into a table of currently open files. Or you can pass it on to your OS routines, if they use a similar unit, channel, or device number concept.

Note that returning an error code in A and Z is NOT an option. It must be done. Suppose there is no way in your FOPEN code to tell if an error occurred. Then always return 0. But don't return garbage. Also FREAD MUST return -1 for end-of-file. FWRITE MUST be able to write small (<256) records, and FREAD MUST return the number of bytes in the record in DE. Tiny-c relies on this data for correct behavior.

This specification is a minimum specification. Your OS probably implements much more. For example, suppose your OS has a file write that can output large arrays. You can use this to implement FWRITE. Tiny-c just won't make use of the full power of your file write. We have tried to make this specification a common denominator to all or most operating systems.

6.1.4 Example of a Tiny-c Cassette File System Interface

Suppose your OS uses a cassette interface, and supports an array write, and a block read with filenames and a choice of recording modes (speeds and/or formats), with these specs:

TWRITE

Parameters

HL points to eight-byte filename
BC points to first byte of array
DE is length of array
A is recording mode (0,1,2,3)

Results

Writes array of (DE) bytes starting at (BC).
Of the modes, 3 is the highest performance mode, 0 the lowest. Writes zero or more full blocks, and one partial block if needed. Starts and stops motor. Calculates and writes checksum. Writes block length in header of each block. Returns no errors, because it assumes recorder is working, etc. If DE is 0, a record of length 0 is written and can be read.

TREAD

Parameters

HL points to eight-byte filename
BC points to first byte of array
A is recording mode

Results

One block is read into memory starting at (BC). DE is set to the number of bytes read. (A) must be the same as when the record was written. Reading stops and A is set to -1 on checksum error. Otherwise a 0 is returned in A.

The problem is to realize AT LEAST the specs of FOPEN, FREAD, FWRITE and FCLOSE using TREAD and TWRITE.

The "T" routines require a mode. The "F" package has no provision for one. The simplest solution is to pick a mode, and always use it. Pick the highest performance mode consistent with your hardware.

The "T" routines have no end-of-file, but FREAD must detect one. We can use the zero length record as an end-of-file. FCLOSE will write such a record, and FREAD will test for this case.

The "T" routines require the filename at read/write time, but the "F" routines don't give it then. So the FOPEN routine will capture the filename in an eight-byte array. In

tiny-c, filenames can be any size. So FOPEN will truncate the name if it is longer than eight bytes, and pad it with blanks if shorter. TREAD and TWRITE then use this captured filename.

The registers and numbers aren't quite the same for the "T" and "F" routines. FWRITE (for example) gives the first byte's address in HL, and the last in DE. TWRITE wants the first in BC, and the array LENGTH in DE. The result code returned in A by TREAD is not what is expected by FREAD. All this can be resolved by some arithmetic and register moves.

The "T" routines don't support multiple files, so unit (in BC in the "F" routines) is ignored. Also at FOPEN time, the read/write switch in DE is ignored.

This example illustrates that implementing the file system interface routines on your computer may not require writing a file management system. Rather, you must carefully plan, and then write, a small amount of code which translates between the tiny-c file system specifications and the file system in your OS.

This example also shows that even though your OS doesn't have the four functions, open, read, write, and close, you can still implement the "F" routines. open and close should be viewed as an opportunity to arrange something, rather than a requirement to do something. Your goal is to realize at least the minimum specifications of the file system.

It is wise to thoroughly test the terminal and file subroutines to see that they conform to the specifications given. Then save them on your mass storage and proceed to Step 2.

6.2 STEP 2 -- Load and Relocate Tiny-c

As furnished, an uninstalled tiny-c has two files:

```
tiny-c
tc.pps
```

Step 2 deals with the first file, tiny-c.

6.2.1 Reading the Tiny-c File

With your machine-readable copy of tiny-c are instructions on how to read it. Follow these instructions until you have a good copy of the file "tiny-c" loaded into memory. Load it at 2000 hex, even if you intend to install it at another origin.

6.2.2 Address Adjustment

As delivered, tiny-c has an origin of 2000 hex. Suppose you intend to install it with an origin of 100 hex. First, load it as described in Section 6.2.1. Then all the addresses in tiny-c must be adjusted by the difference between 2000H and 100H, namely -1F00H.

The program in Figure 6-1 does this.

Some three-byte instructions have data, not addresses. For example:

```
LXI      D,-C
LXI      H, 0
```

In tiny-c these data are always of small magnitude. Addresses are always of the form 2XXX or 3XXX (hex). So the relocation program tests for this, and does not adjust data.

First, load tiny-c at 2000H. Then load the ADJADDR program including its subroutines and data tables. Load this anywhere outside the tiny-c area (2000-3100 hex). It is shown compiled at 6000H. If this area is available, use ADJADDR as compiled in Figure 6-1. Otherwise, its 20 addresses can be relocated by hand. Then ADJADDR can be loaded somewhere else.

Set BC to the adjustment: -1F00H = E100H. In general it is:

yourorigin - 2000H.

Start execution at ADJADDR. When the computer halts, the addresses have been adjusted.

Examine memory to see if the addresses make sense. The listings of tiny-c are in Appendix A. Is the first address correctly relocated? Is the last? If so the rest are probably correct.

Record the results (2000-3100 hex) and reload them at your origin. Examine the addresses again to see if they make sense.

FIGURE 6-1

6000			0000	; Adjusts the address of tiny-c, version X.31. The table
6000			0010	; TCADDS separates data from program areas. However
6000			0020	; certain data (like error codes) which fortuitously
6000			0030	; look like one-byte 8080 instructions are not shown
6000			0040	; in the table. So if you change the error codes, the
6000			0050	; table must be changed.
6000			0060	;
6000	21	06	0070	ADJADDR LXI H, TCADDS
6003	C3	80	0080	JMP ADJUST
6006	00	20	0090	TCADDS DW 2000H
6008	08	20	0100	DW 2008H
600A	2B	20	0110	DW 202BH
600C	33	20	0120	DW 2033H
600E	C0	20	0130	DW 20C0H
6010	C5	23	0140	DW 23C5H
6012	C7	23	0150	DW 23C7H
6014	04	24	0160	DW 2404H
6016	0F	24	0170	DW 240FH
6018	D1	24	0180	DW 24D1H
601A	E1	24	0190	DW 24E1H
601C	68	25	0200	DW 2568H
601E	F3	27	0210	DW 27F3H
6020	27	28	0220	DW 2827H
6022	2C	28	0230	DW 282CH
6024	88	28	0240	DW 2888H
6026	8F	28	0250	DW 288FH
6028	1E	2A	0260	DW 2A1EH
602A	25	2A	0270	DW 2A25H
602C	7A	2A	0280	DW 2A7AH
602E	80	2A	0290	DW 2A80H
6030	3A	2C	0300	DW 2C3AH
6032	3C	2C	0310	DW 2C3CH
6034	FF	2C	0320	DW 2CFFH

6036	12	2D	0330	DW	2D12H	
6038	97	2D	0340	DW	2D97H	
603A	C0	2D	0350	DW	2DC0H	
603C	09	2E	0360	DW	2E09H	
603E	48	2E	0370	DW	2E48H	
6040	86	2E	0380	DW	2E86H	
6042	89	2E	0390	DW	2E89H	
6044	DF 30	F8 30	0400	DW	30DFH 30F8H	
6046	6B	25	0410	DW	256BH	
6048	F1	27	0420	DW	27F1H	
604A	00	00	0430	DW	0	;end of table
604C			0440	D5 DS	52	;to expand the table
6080			0630			
6080			0640			
6080			0650			
6080			0660			
6080			0670			
6080			0680			
6080			0690			
6080			0700			
6080			0710			
6080			0720			
6080	5E		0730	ADJUST	MOV	E,M ;from address
6081	23		0740		INX	H
6082	56		0750		MOV	D,M
6083	23		0760		INX	H
6084	7B		0770		MOV	A,E ;test if done
6085	B2		0780		ORA	D
6086	CA	96 60	0790		JZ	HALT
6089	D5		0800		PUSH	D
608A	5E		0810		MOV	E,M ;to address
608B	23		0820		INX	H
608C	56		0830		MOV	D,M
608D	23		0840		INX	H
608E	E3		0850		XTHL	;from -> HL, pointer -> stack
608F	CD	9A 60	0860		CALL	ADJFRT0 ;adjust this range
6092	E1		0870		POP	H ;pointer -> HL


```

6093 C3 80 60
6096 76
6097 C3 96 60
609A
609A
609A
609A 7B
609B 95
609C 7A
609D 9C
609E D8
609F CD CA 60
60A2 FE 01
60A4 CA BD 60
60A7 FE 02
60A9 CA BC 60
60AC 23
60AD 23
60AE 7E
60AF FE 20
60B1 DA BD 60
60B4 FE 40
60B6 DC C1 60
60B9 C3 BD 60
60BC 23
60BD 23
60BE C3 9A 60
60C1
60C1 2B
60C2 79
60C3 86
60C4 77
60C5 23
60C6 78
60C7 8E
60C8 77
60C9 C9

```

```

0880      JMP      ADJUST
0890  HALT      HLT
0900      JMP      HALT
0910      ;Adjusts address of instructions from (HL)...(DE)
0920      ; inclusive by adding (BE).  An address is modified
0930      ; only if it is from 2000 to 3FFF inclusive.
0940  ADJFRT0 MOV     A,E      ;done if HL > DE
0950      SUB     L
0960      MOV     A,D
0970      SBB     H
0980      RC
0990      CALL    INSIZE      ;Get size of instruction at
1000      CPI     1          ; (HL)
1010      JZ      SKIPONE
1020      CPI     2
1030      JZ      SKIPTWO
1040      INX     H          ;3 byter, point to hi byte
1050      INX     H          ; of its address
1060      MOV     A,M        ;test for page 2x or 3x
1070      CPI     20H
1080      JC      SKIPONE
1090      CPI     40H
1100      CC      ADJONE      ;adjust this one
1110      JMP     SKIPONE
1120  SKIPTWO INX     H
1130  SKIPONE INX     H
1140      JMP     ADJFRT0
1150      ;Adjusts one address.  (HL) points to its hi byte.
1160  ADJONE  DCX     H          ;lo byte
1170      MOV     A,C
1180      ADD     M
1190      MOV     M,A
1200      INX     H          ;hi byte
1210      MOV     A,B
1220      ADC     M
1230      MOV     M,A
1240      RET

```


60CA		1250	;Determine size of instruction at (HL). Restores all	
60CA		1260	; registers except A, where it returns a 1, 2, or 3.	
60CA E5		1270	INSIZE	PUSH H
60CB 7E		1280		MOV A,M ;inst byte
60CC 21 00 61		1290		LXI H,THREEBT
60CF CD EA 60		1300		CALL LOOKUP
60D2 CA D9 60		1310		JZ IN2
60D5 3E 03		1320		MVI A,3 ;is three bytes
60D7 E1		1330		POP H
60D8 C9		1340		RET
60D9 21 17 61		1350	IN2	LXI H,TWOBT
60DC CD EA 60		1360		CALL LOOKUP
60DF CA E6 60		1370		JZ INONE
60E2 3E 02		1380		MVI A,2 ;is two bytes
60E4 E1		1390		POP H
60E5 C9		1400		RET
60E6 3E 01		1410	INONE	MVI A,1 ;is one byte
60E8 E1		1420		POP H
60E9 C9		1430		RET
60EA		1440	;One-byte table lookup subroutine. Byte is in A.	
60EA		1450	; (HL) points to beginning of table, ended with a	
60EA		1460	; null byte. Returns NZ on found, Z on found. All	
60EA		1470	; registers, including A, are restored.	
60EA E5		1480	LOOKUP	PUSH H ;only flags can be changed.
60EB C5		1490		PUSH B
60EC 47		1500		MOV B,A ;lookup byte -> B
60ED 7E		1510	LK2	MOV A,M ;next byte of table
60EE B7		1520		ORA A
60EF CA FC 60		1530		JZ LK4 ;end of table, byte not found
60F2 90		1540		SUB B
60F3 CA FA 60		1550		JZ LK3 ;found
60F6 23		1560		INX H
60F7 C3 ED 60		1570		JMP LK2
60FA DE 01		1580	LK3	SBI 1 ;A has 0, so this sets NZ
60FC 78		1590	LK4	MOV A,B ;this restores A without
60FD C1		1600		POP B ; affecting flags
60FE E1		1610		POP H

60FF	C9	1620	RET		
6100	C3	1630	THREEBT	DB	0C3H
6101	C2	1640		DB	0C2H
6102	CA	1650		DB	0CAH
6103	D2	1660		DB	0D2H
6104	DA	1670		DB	0DAH
6105	F2	1680		DB	0F2H
6106	FA	1690		DB	0FAH
6107	CD	1700		DB	0CDH ;calls
6108	C4	1710		DB	0C4H
6109	CC	1720		DB	0CCH
610A	D4	1730		DB	0D4H
610B	DC	1740		DB	0DCH
610C	F4	1750		DB	0F4H
610D	FC	1760		DB	0FCH
610E	22	1770		DB	022H ;xHLD
610F	2A	1780		DB	02AH
6110	32	1790		DB	032H ;xDA
6111	3A	1800		DB	03AH
6112	01	1810		DB	01H ;LXI
6113	11	1820		DB	11H
6114	21	1830		DB	21H
6115	31	1840		DB	31H
6116	00	1850		DB	0 ;end of three-byters
6117	06	1860	TWOBT	DB	06H ;MVI
6118	0E	1870		DB	0EH
6119	16	1880		DB	16H
611A	1E	1890		DB	1EH
611B	26	1900		DB	26H
611C	2E	1910		DB	2EH
611D	36	1920		DB	36H
611E	3E	1930		DB	3EH
611F	C6	1940		DB	0C6H ;xyI
6120	CE	1950		DB	0CEH
6121	D6	1960		DB	0D6H
6122	DE	1970		DB	0DEH
6123	E6	1980		DB	0E6H

6124	EE	1990	DB	0EEH	
6125	F6	2000	DB	0F6H	
6126	FE	2010	DB	0FEH	
6127	DB	2020	DB	0DBH	;in/out
6128	D3	2030	DB	0D3H	
6129	00	2040	DB	0	;end of two-byters

End of FIGURE 6-1

6.3 STEP 3 -- Load the Interface Routines

In Section 6.1.4 you saved these routines on your mass storage. Now reload them, if they are not still in memory.

6.4 STEP 4 -- Link Tiny-c to the Interface Routines

The tiny-c transfer vector starts at ORG+0A hex. There are seven JMP statements. Fill their address fields with the addresses of the seven interface routines:

ORG+0A	JMP	INCH
ORG+0D	JMP	OUTCH
ORG+10	JMP	CHRDY
ORG+13	JMP	FOPEN
ORG+16	JMP	FREAD
ORG+19	JMP	FWRITE
ORG+1C	JMP	FCLOSE

Note that the addresses given are for the JMP instruction; C3 hex must be in ORG+0A, ORG+0D, ORG+10 ... The next two bytes are the address field.

6.5 STEP 5 -- Modify the Other Installation Vector Elements

Sections 6.5.1, 2, and 3 describe those additional entries in the installation vector that are essential to an installation. There are several optional entries that give additional facilities. These are described in Section 6.5.4.

6.5.1 Upper Case Option

Tiny-c is at its best with a full ASCII keyboard and display. As delivered, it assumes full upper and lower case operation. In particular, the keywords in the keyword table are all lower case. When you use lower case keywords, you

will see it makes programs more readable, more pleasant to the eye.

If you have full upper/lower case for both your keyboard and display, skip this subsection.

If your keyboard is upper case only, then your tiny-c programs will have upper case keywords ("WHILE" instead of "while", etc.). There is one situation to watch out for: if your keyboard happens to be full upper/lower, but your display is upper case only, it may seem safe to leave tiny-c in its lower case mode. The program text is entered in lower case, it runs in lower case, and it is stored in lower case. Only the display maps it to upper case, and that is readable enough. But, what if you accidentally enter "while"? The program won't run. When you display it, it reads WHILE, giving you no clue to the problem. We recommend two modifications if you have this kind of terminal:

1. Implement the upper case mod, given below.
2. Also implement case fold logic in INCH. Lower case alphabetics arriving from the keyboard should be mapped to upper case by software in INCH. (See Section 6.1.1.)

This will give you a quality interface to the terminal.

To implement upper case:

The table of keywords is located at ORG+63 hex through ORG+93 hex. Modify NONZERO bytes of this table by subtracting hex 20. So 69H becomes 49H. Leave the zero bytes as 0. Starting at ORG+0D09H are four bytes that must be similarly adjusted. Finally, all alphabetics in the file "tc.pps" must be case-folded to upper case. This must be carefully done so that non-alphabetics are not modified.

6.5.2 Allocation of Memory

After tiny-c and the interface routines and operating system routines are loaded, you should have a large block of memory left over. This must be allocated into four working areas for tiny-c. They are:

	Recommended Allotment
STACK	128 bytes (decimal)
FUN	128 bytes (decimal)
VAR	15% of the remainder
PR	all the rest

The STACK is for tiny-c calculations, and should not be confused with the 8080 stack. The space allotments are for "average" programs, and can be rounded to convenient numbers. Assign the beginning addresses for the blocks, and enter these values in the locations:

BSTACK	ORG+36H
BFUN	ORG+3AH
BVAR	ORG+3EH
BPR	ORG+42H

Calculate the end addresses, and put their NEGATIVE value in the locations:

ESTACK	ORG+38H
EFUN	ORG+3CH
EVAR	ORG+40H
EPR	ORG+44H

EXAMPLE: Assume an origin of 0. The area to be allocated is from 1200H to 3FFFH. There are

$$3FFF - 1200 = 2DFH = 11775 \text{ dec}$$

bytes available. Allocate 128 for STACK and 128 for FUN. That leaves 11519. 15% of this is 1728 for VAR, leaving 9791 for PR. We will round these to convenient hex quantities.

	allot dec	rounded allot hex	begin	end+1	-end-1
	=====	=====	=====	=====	=====
STACK	128	80	1200	1280	ED80
FUN	128	80	1280	1300	ED00
VAR	1728	700	1300	1A00	E600
PR	9791	2600	1A00	4000	C000

End+1 is calculated as begin + rounded allotment. This is the next item's begin. The boxed quantities are entered as the B and E addresses for STACK, FUN, VAR and PR respectively.

	address*	contents (all hex)
	=====	=====
BSTACK	36	1200
ESTACK	38	ED80
BFUN	3A	1280
EFUN	3C	ED00
BVAR	3E	1300
EVAR	40	E600
BPR	42	1A00
EPR	44	C000

*origin 0 assumed. Otherwise add
your origin to the addresses shown.

Note that all addresses must actually be entered in the usual 8080 byte-reversed order. This example is NOT in that order to make the contents readable and the principles clear.

Anytime BPR is changed, PPS must be reloaded as described in Section 6.8, and the load-and-go tiny-c re-recorded as described in Section 6.9.

6.5.3 8080 Stack

If your operating system allocates an 8080 stack, then the two bytes starting at ORG+46H, must have the value 0. (The delivered tiny-c has this value.) If your system will start up tiny-c without having allocated an 8080 stack, then tiny-c will allocate it for you on a COLD START. (All tests through Section 6.8 use cold starts. Section 6.9 explains hot starts.) Determine a block of memory to use. Put its HIGHEST address into ORG+46H. Use the usual 8080 byte-reversed order.

EXAMPLE: The block from 1000H to 11FFH will be used for the stack, and tiny-c will allocate it. (The origin is 0.)

	address	contents (NOT shown byte-
	=====	===== reversed)
S8080	46H	11FFH

6.5.4 Optional Entries in the Installation Vector

6.5.4.1 User Machine Call Jump Address

If you program a user Machine Call you must allocate some memory for its code. Do this by carefully accounting for all the memory space used:

1. by tiny-c from ORG through ORG+1100.
2. by your own installation code.
3. in allocating the four working areas (Section 6.5.2) and possibly the 8080 stack (Section 6.5.3).

Items 1 and 2 are probably bolted down in certain addresses of memory and not easily changed. Item 3 is easily changed. So by changing item 3 allocations you can make room for code for private (user) Machine Call code. Note that changing the allocations means PPS must be reloaded (see Section 6.8).

Load the Machine Call program in the space freed up. Now link it to tiny-c by putting an address in the jump instruction at ORG+1F hex:

(ORG+1F) C3 XX XX <--- fill in this address

Note that this jump instruction already has an address (not 0000). If you have an uninstalled tiny-c, it jumps to an error routine called MCESET, (see Section 2.10). Check your installation document if your tiny-c is installed, since it may already have private MCs. If so, the JMP at ORG+1F is already patched to the correct address and should not be changed. Your installation document will tell you:

1. What MC numbers are already used, and which you may use for additional MCs.
2. How to patch them in.

Finally, capture all this work by doing the procedure in Section 6.9.

6.5.4.2 Choice of Escape Character

You may halt an application by typing ASCII ESC. If this choice of character is unsuitable for your terminal it can be changed. At ORG+35 hex is the character which, when typed, causes an application halt. You can change it to any value not needed for programming or for data.

6.5.4.3 Statement Control Opportunities

Tiny-c will CALL external subroutines at three points:

- When any program is begun,
including PPS and applications.
- At the start of each statement.
- When any program completes,
including PPS and applications.

(Note that by "program" we do not mean function. We mean a main program, i.e., all of PPS, or all of an application.)

There is room for three jumps in the installation vector at ORG+22, ORG+25, and ORG+28 hex. As furnished, each has the three one-byte instructions:

```
NOP
NOP
RET
```

These instructions do nothing except assure the continued running of tiny-c.

The opportunity to link to external subroutines is provided for several purposes; for example, to create

1. a debugger, with single step
and variable display capability,
and even breakpoints.
2. a profiler, which counts how
often each statement gets
executed.

Use of this capability requires knowledge of Chapter V, plus study of the listings in Appendix A, plus imagination.

6.6 STEP 6 (and 6') -- Write to Disk Or Tape

Step 5 concluded the installation of raw tiny-c. This should be written to your mass storage before testing. If your OS allows, set it up so loading will automatically cause execution to begin at ORG.

Step 6' is simply the converse of 6. Make sure this can be done successfully before going to Step 7.

6.7 STEP 7 -- Test the Results So Far

Load tiny-c and start it at ORG. The prompter

>>>

should appear. This indicates you are in the loader.

If the prompter does not appear, check that your OUTCH routine is loaded, linked properly at ORG+0D, and working properly. If you get one > and then either garbage or nothing, check that your OUTCH restores register A. Finally, trace the code in Appendix A from ORG to the first CALL INCH in the subroutine LOADER. Check that your loaded tiny-c has good bytes on the path.

The loader has three commands:

```
.r filename
.g
.x
```

The first reads a file containing a tiny-c program or part of a program. The .r command can be used more than once if a program has parts stored in several files.

The second command, .g, causes the load program to start at the function "main". First, the entire program is linked (see Section 5.9.11). Several errors can be detected here, indicating a bad tiny-c program or an improperly functioning file read routine (see Section 6.1.2).

The third command, .x, causes a jump to location 0, which usually returns control to your OS. The jump instruction (actually CA hex, Jump on Zero) is located at ORG+10F1 hex, and can be given another address if needed.

6.7.1 Null Program Test

The first test is to give the .g command, i.e., try to run a null program. The diagnostic

```
DONE 26 aaaaa
```

will be printed. aaaaa is the address where the program looked for "main". Its value is BPR+5, and it is printed in decimal.

6.7.2 Simple Program Test, Hand Loaded

In Section 6.5.2, memory was allocated. A value for the beginning of program space (BPR) was determined. The example derived 1A00 hex for BPR. In this test we will hand load 14 bytes, a very small tiny-c program, into this space. We will set a pointer to the first byte BEYOND the program. And we will run the program.

Load tiny-c and do a cold start (i.e., start it at ORG). When the >>> appears, interrupt it using your operating system's usual method. Then, starting at BPR+9, put these bytes into memory:

address =====	ASCII value =====	hex value =====
BPR+9	m	6D
BPR+A	a	61
BPR+B	i	69
.	n	6E
.	[	5B
.	M	4D
	C	43
	'	27
	x	78
	'	27
	,	2C
	l	31
	]	5D
BPR+16 hex	null	00

Now (in hex) calculate the value of $-(BPR+16)$. Put that value in $ORG+60$ and $ORG+61$ in the usual 8080 byte-reversed order. For example, if BPR is 1A00 hex, then

```

-(BPR + 16)
= -1A16
= E5E9 + 1    (hex complement + 1)
= E5EA

```

Put EA into 2060, E5 into 2061, i.e., low-order byte first.

You have now loaded a program. Be sure the 8080 stack is exactly the way it was when you interrupted tiny-c. Warm start tiny-c by starting it at $ORG + 3$. (If you start it at ORG , the loaded program gets erased.) The triple prompter >>> should appear. Give the command .g. The copyright message should be printed, followed by the single character 'x' on a line by itself. Next, DONE should appear, also on a line by itself. Then the triple prompter >>>. If all this happens, you have just run your first tiny-c program.

If it doesn't work don't look in your interface routines for the problem. Only your OUTCH program was used by this test, and that had to work to produce the >>>. Most likely the problem is in the setup of the test. Go over it carefully. The series BPR through BPR+8 should contain

```
[main();]
```

Does it? If not, then you forgot the cold start. If it does, check that you entered the 14 bytes correctly and calculated the negative of BPR+16 correctly. Did you use upper case for MC? It must be upper case. The bytes "main" at BPR+1, ..., BPR+4 must be exactly the same as at BPR+9, ..., BPR+C. They can be upper or lower case, but must be the same. Be sure you did not confuse BPR (the address of the program space) with ORG+42, a word in which that address must be stored.

6.7.3 Simple Program Test, File Loaded

Prepare a file with the same 14-byte program as in Section 6.7.2. Make sure the file is readable by your FREAD routine (see Section 6.1.2), and that it returns decimal 14 (hex 0E) in DE as the number of bytes read. Make sure that a second call to FREAD returns an end-of-file signal.

Now cold start tiny-c (start it at ORG). Give .r filename. The file should read into BPR+9, ..., BPR+C.

Interrupt and examine the program buffer. Its first nine bytes should be [main();]. Its next 14 bytes should be the test program. Examine ORG+60. The same end pointer you calculated for Section 6.7.2 should be there.

Resume execution of tiny-c (making sure the 8080 stack is unmodified). Give the .g command. The program should run. If it doesn't, make all the checks given in Section 6.7.2.

6.8 STEP 8 -- Prepare PPS for Loading

If you intend to use an external editor to prepare tiny-c programs, then your installation is complete. The external editor must, of course, write files compatible with your

FREAD program. And every program you prepare must have a main program called "main".

If you intend to use PPS, you must complete Step 8. PPS is a tiny-c program. You must prepare it for reading, read it, and start it just as you did the sample program in Section 6.7.3.

First PPS must be written on a file compatible with your FREAD program. PPS is the second file of your machine-readable media. It is also shown in Appendix C. With your machine-readable copy are instructions on how to read the second file. Study these until you understand them.

Write a program that reads all of PPS into memory, then (using your FWRITE and FCLOSE) writes it to your own media in your own format. Run this program. Check to see that you have a good copy. If you have a "file-examine" capability, use it to make sure the output file has a good copy of PPS, and will be readable by your FREAD. In any case, write a program using your FOPEN and FREAD to read the file. Does it return the correct number of bytes in DE for each record read? Does it detect an end-of-file correctly?

When you are sure your copy is correct, cold start tiny-c and read PPS using .r filename. Interrupt tiny-c and examine memory to see if a good copy of PPS is in BPR+9, ... Find its end. There should be a null byte (hex 00) at the very end. Calculate the negative of the address of this null byte. That value should be in ORG+60. If you find a problem, it's probably because this is the first multi-record program loaded. Check that FREAD returns 100 hex for 256-byte records. If everything seems in order, start PPS with .g.

The tiny-c copyright message should print. Then, on a line by itself, the single prompter

>

will appear. This was produced by a tiny-c statement in PPS!

If a > does not appear, a DONE message should appear:

DONE ee aaaaa

where ee is a decimal error code, and aaaaa is a DECIMAL (not hex) address. This is a tiny-c diagnostic, error ee, at address aaaaa. (The error codes are in Section 3.5.) Convert

the address to hexadecimal, and examine memory there to find the problem.

6.8.1 Possibly Required PPS Changes

The block size of the file management system was determined by the installer in Step 1 of the installation process. That size, less one, is coded into the writefile function of PPS in the statement

```
if(1>255)1 = 255
```

(This is the statement that implements the "portability view" for PPS. See Section 2.9.2). writefile is about one-third from the top of PPS, just before the endlibrary statement.

These two integers are encoded in ASCII, so the line appears in hex as:

```
69 66 28 6C 3E 32 35 35 20 29 6C 3D 32 35 35 20 0D
i f ( 1 > 2 5 5 sp ) 1 = 2 5 5 sp ret
```

If the design block size is not 256, then the size-less-one must be put into this statement, in two separate places. The space (20 hex) at the end of each such place allows up to four digits for this.

6.8.2 Optional PPS Changes

The ASCII character CAN (meaning cancel, hex 18) is the "line kill" character. This is encoded as the decimal number 24 in the function gs, near the beginning of PPS (See Appendix C). The digits of the number are encoded in two ASCII bytes: 32 34 hex. Similarly, the ASCII character DEL (hex 7F) is the "character kill". It is encoded as decimal 127, and is written as three ASCII bytes: 31 32 37 hex. These can be changed to suit your habits, or the limitations of your keyboard.

6.9 Write the "Load-And-Go" Tiny-c to Mass Storage

You can always go through the three (or four) step sequence:

```
load tiny-c
(start at ORG)
>.r tc.pps
>.g
```

to start the Program Preparation System. It is more convenient to do a single load with an auto start, if you have one. To prepare for this, write out all of tiny-c, the interface routines, any system routines they need, and 4600 bytes starting at EPR (the tc.pps text) as one file on your mass storage. Arrange an automatic execution at ORG+6, if you have a way of doing this. This is a "hot start" which starts not only tiny-c, but any system program it has loaded. In this case, the Program Preparation System will be started. Now loading (and starting) this file gets you going in only one or two steps, because the .r and .g steps are "automated".

If you implemented the stack allocation option (Section 6.5.3) and want to do a hot start, then put this program in your installation code:

```
      HOTSTART      LHLD      ORG+46
                      SPHL
                      JMP      ORG+6
```

Then HOTSTART is your hot start address.

6.10 If Things Go Wrong

It should be clear from reading this that to install tiny-c, you must, at least, know how to use a file system and your i/o interface routines. You might even have needed to write a file system. So you're not a software novice.

If things blow up, review your i/o interface routines. Do they work as required? Test them again. Are they correctly linked? Do you have an 8080 stack? Are the addresses correctly relocated?

A crucial step is in Section 6.7 when you start tiny-c at ORG. This is the first attempt to execute tiny-c. Trace the code in Appendix A. It is a short sequence from ORG to the first CALL INCH in subroutine LOADER. If you get there, and if INCH works, you should be able to type in a line of text. Type a few characters. Then interrupt and examine BUFF. If characters are going into BUFF, and are in the right case, and if the first two are ".r", then a carriage return should cause a call on FOPEN.

Are you getting there? If so, is a 0 returned in A? Is the Z flag set? The JNZ LOADER after the CALL FOPEN should not branch! If it does you have an FOPEN problem. Next, FREAD should get called. It should return 0 with Z set on the first call. Is that happening? If so, you've read one block of tc.pps. Check BPR (ORG+42 hex). Is it where you want tiny-c programs loaded? If so, go look there and see if a block of tc.pps is really there. If it isn't, you have FREAD problems.

Let the read loop (L2) run to completion. An end-of-file should be detected by FREAD, a -1 returned, and Z not set. This causes a jump back to LOADER. Now interrupt and examine the program area. Is all of tc.pps there? Are there gaps? Are there missing bytes? If so, FREAD is not returning correct byte counts in DE. Is PROGEND pointing to the last byte of tc.pps? (PROGEND is stored negative. Write out its 16 bits. Then complement them. Then add one. That address should contain the last byte of tc.pps, which is a null byte after the last carriage return.)

If all this is correct, then the go command ".g" should cause LOADER to return to the location HOT. From then on, you're in tiny-c code, except for calls to INCH and OUTCH. If it still doesn't work, then print a hex dump of ORG through ORG+1100 hex. Sit down with a good friend and read every byte, and compare with the listings in Appendix A.

6.11 Key Points of a Tiny-c Installation

In this documentation several references are made to installer-determined parameters. Any tiny-c installation should document the following key points:

1. The Storage Map parameters. These are:

ORG

Where private MCs can start

Where space starts

2. Private MCs furnished. These should be documented. We recommend you edit the standard library to incorporate these MCs with descriptive names, as described in Section 2.9.
3. Line kill and character kill characters, as described in Section 6.7.2. State what they are, even if not changed.

Similarly, define the application halt (ESCAPE) key chosen in Section 6.5.4.

4. Design decisions on the interface routines:

- whether or not filenames are used, and if so, how long they can be.
- whether or not filesize is relevant in FOPEN.
- whether or not multiple units are supported, and restrictions if they are.
- whether or not read/write access modes are required.
- block size.

5. Files on the machine-readable copy, their names, recording modes, and software to load them.

6. Other installation options:

- upper case, or upper/lower case;
- allocation of memory;
- how the 8080 stack is allocated and where.

7. PPS options, particularly the size of program space. It is also convenient to know:

-- the exact bytes to modify for larger or smaller program space. These are the ASCII encoded decimal digits DDDD in the PPS text:

```
char ln(64), pr(DDDD)
```

and four lines later

```
lpr = DDDD
```

-- The exact byte address which is line zero of an application.

8. We recommend that an emergency utility be built to save your work in case of a system crash. Assuming your operating system allows you to browse through memory looking at the software corpse, it is not unlikely you can find the first and last bytes of your valuable but not yet recorded work. The utility accepts two addresses, and records everything from the first to the second (inclusive) as a file with a standard filename.

6.12 Modifying the Program Preparation System

The PPS can be used to edit itself. PPS is stored in a file called "tc.pps". This is the file produced by Step 8 of the installation procedure (Section 6.8). Read it in by using the .r command, edit it, and write the edited version using .w. Start it and test it with .main (the name of the main function is "main"). In this way you can

-- add functions to the standard library, and
-- add or modify features of PPS.

In the middle of the file is an "endlibrary" statement. This unique statement separates library symbols from global symbols. All global symbols that occur before the endlibrary statement are library symbols, and can be accessed by both

PPS and applications. All that occur after the endlibrary are globals to PPS, and are not accessible to applications. There must be one and only one endlibrary statement.

Once you are satisfied that your new version of PPS is working, go to Step 9 (Section 6.9) and re-record tiny-c with an imbedded new PPS.

6.12.1 The System Loader

To load an edited PPS, interrupt or halt your 8080. If the tiny-c interpreter is not loaded, then load it. Start it at ORG (See Section 6.2.2). This is a cold start. It erases any PPS already loaded. Then the system loader is entered. The prompter:

>>>

is issued. The system loader accepts three commands:

>>>.r filename

Reads a file from cassette or floppy disk, appending it to what was read previously. [Note: Some installations of tiny-c may not use the filename.] The set of files read using this command becomes the system level program (level 0 as defined in Section 4.3.4).

>>>.g

The system level program is started at the function named "main". main has no arguments.

>>>.x

A jump is made to the OS. See Section 6.7 for how this jump is done.

NOTE: You may use .r repeatedly to load more than one file. In this case library routines must be loaded first, and the system programs last. Within the entire set of files must be one and only one endlibrary statement partitioning the library from the system programs.

6.12.2 Debugging a New PPS

There are no editing tools in the system loader. The debugging aid is a single diagnostic: either

DONE

which means the function main completed without error; or

DONE ee aaaaa

indicating error number ee occurred at byte address aaaaa.

The error codes are listed in Section 3.5. The address is given in decimal, and is the memory address of the byte being interpreted when the error was detected. This is the sole clue to a problem in a new PPS. For this reason, a new or modified PPS should be debugged while operating as an application under the standard PPS, which gives better clues and allows online editing.

Tiny-c "Load and Go" Version

for the

Poly 88 (Monitor 4.0)

(April, 1978)

Note: This is not a tiny-c manual. This is a
supplement to the tiny-c OWNER'S MANUAL,
which is available separately.

Copyright, 1978, Tiny-c Associates

Contents

Section	Page
P1 Load and Start, upper/lower case keyboards	2
P1A Load and Start, upper case only keyboards	2
P2 Special Characters	4
P3 Mods to PPS	4
P4 Additional Library Functions	4
P5 Additional Private MC's	6
P6 Principal Memory Addresses	6
P7 Utilities Provided	7
P8 Reallocating Memory	9
P9 File System Error Codes	12
P10 Reading Byte Mode Files	12
Appendix	After page
8080 Installation Source Code	12

Note: This is not a tiny-c manual. This is a
supplement to the tiny-c OWNER'S MANUAL,
which is available separately.

P1. Load and Start (upper/lower case keyboards)

The tape is recorded in Byte standard, with the standard Poly 88 tape format. It has a leader of about 60 seconds, then the file TC, then a 30 second inter-file gap, then the file TC.PPS. Power up the computer (or push reset) and type (in upper case):

B
TC

Tiny-c will load. At the end of loading, tiny-c auto-starts at 2000H. The tiny-c copyright is displayed, and a triple prompter:

>>>

Now you can load the PPS. Type:

.r TC.PPS

You must type the period; and r (for read) must be lower case. There must be one blank, then (in capitals) TC.PPS; and finally a carriage return. Position the cassette to read the TC.PPS file, and start it in PLAY mode. When TC.PPS has been read, the triple prompter will appear again. Type:

.g

This will start execution of the PPS. It prints the copyright message again, then a single prompter:

>

Chapter 3 of the OWNER'S MANUAL tells you how to use the PPS.

As provided, tiny-c and the PPS are recorded in BYTE standard, and therefore load very slowly. In Section P7, a utility called TCMOD is described with which you can re-record tiny-c in polyphase, and in a single file with an auto-go record; thus both tiny-c and the PPS can be loaded quickly in a single transaction, and will auto-start.

P1A. Load and Start (uppercase only keyboards)

Follow the instructions above until the first >>> appears. Then follow the instructions in Section 6.5.1 of the OWNER'S MANUAL. Record the modified TC using the Small Memory Dumper, or the TCMOD utility described in Section P7 of these instructions.

Modifying PPS to upper case is harder. It is recorded as data records, so cannot be read by monitor 4.0. Load the modified TC and start it at 2000H.

SPJ2000
G

A triple prompter will appear:

>>>

type:

.R TC.PPS

using a capital R, but otherwise as described in Section P1. When PPS has been read, the triple prompter will appear again.

You can try to start it, but it won't work. Type .G and see what happens. (No damage will be done.) Your modified TC only recognizes upper case keywords, like WHILE, but PPS is all lower case, like while. So they don't match. But you have PPS in RAM, so you can work on it.

Go into front panel mode and find PPS's first and last bytes. See Section P6 for where they should be and how to recognize them. Put the address of the first byte into HL, and the last byte into DE. (If you have not changed BPR, then HL should be 3D89, and DE 508E.) Then enter this program at 5800H.

5800	7E	LOOP	MOV	A,M	
5801	FE 60		CPI	60H	;NO CARRY MEANS ALPHA
5803	DA 09 58		JC	NEXT	
5806	D6 20		SUI	20H	;CONVERT TO CAP
5808	77		MOV	M,A	;PUT IT BACK
5809	23	NEXT	INX	H	;BUMP ADDRESS
580A	7A		MOV	A,D	;TEST FOR END
580B	BC		CMP	H	; IF DE == HL THEN
580C	C2 00 58		JNZ	LOOP	; DONE
580F	7B		MOV	A,E	
5810	BD		CMP	L	
5811	C2 00 58		JNZ	LOOP	
5814	76		HLT		;ALL DONE.
5815	C3 14 58		JMP	\$-4	;IN CASE OF INTERRUPTS

Proof-read it. Then start it at 5800H. Now your PPS is all upper case. Use TCMOD (section P7) to capture all your work. Also the other two utilities should prove useful.

You can print the modified PPS using PUTIL and verify it is indeed in upper case. And you can record PPS in its upper case version using DDUMP. This will be useful if you ever buy more memory and want to reallocate memory, or if you want to edit PPS.

P2. Special characters

The line and character kill characters, and the program halt characters are unchanged from the tiny-c standard. They are:

<u>ASCII name</u>	<u>Hex code</u>	<u>Function</u>
DEL	7F	Delete one character
CAN	18	Cancel a line
ESC	1B	Halt application program

CAN is control-X on many keyboards. DEL is sometimes called RUBOUT. ESC (for escape) is usually a separate key labeled ESC. If not, find out what key on your keyboard generates 1B hex. It may be a control-key combination.

P3. Mods to PPS

Two new commands are included in PPS for interfacing to a printer. They are:

.pon	turns the printer on
.poff	turns the printer off

As delivered, the USART is programmed for 300 baud with 1 start, 2 stop, and no parity bits. This is compatible with TI700 devices, DECWRITER, and many other 300 baud printers. Instructions for changing the USART program for other speeds or framing bits are given in section P7.

P4. Additional Library Functions

clear

Issues an ASCII Form Feed, thus clearing the screen, and homing the cursor.

home

Issues an ASCII Vertical Tab, thus homing the cursor.

plot int row, col, onoff

This function is defined in Section 4.6 of the tiny-c OWNER'S MANUAL.

plotel int row, col, off
char el(0)

Plots several points in one call, so an entire "picture element" is painted at high speed. El is an array of relative (row, column) pairs in the form:

rr1, rcl, rr2, rc2,....rrn, rcn, 128

Note that the list has an odd number of elements, and a 128 signals its end. The elements at (row + rr1, col + rcl), (row + rr2, col + rc2)....(row + rrn, col + rcn) are either painted (onoff!=0) or erased (onoff==0). The number of points offscreen is returned. So if 0 is returned, the entire picture element is onscreen. Here is an example use of plotel to draw this picture element:

	-2,1	
-1,0		
0,0	0,1	0,2

The origin of the element (square 0,0) is picked arbitrarily. The others are coded relative to the origin.

```
char el(10)      /* declare el and make
el(3) = el(9)=1  /* its contents be
el(5) = 2        /* 0,0, 0,1, 0,2 -1,0, -2,1,
el(6) = -1       /* 128
el(8) = -2
el(10) = 128
call plotel 24, 64, 1, el
                /* draw the picture
                /* element near the middle of
                /* the screen.
```

printon

Programs the USART for 300 baud, 1 start, 2 stop, and no parity bits for device 1 (the printer) and leaves it idling. This is compatible with the DECWRITER, TI700 series, and many other 300 baud printers. Instructions for changing the baud rate are in Section P7. Also sets up the interrupt handler SRA4 so that any byte transmitted through WH1 goes to both the screen and the printer. Tabs are changed to an appropriate number of blanks. Carriage returns are padded with a line feed and four rubouts.

printoff

Turns the USART off and restores the interrupt handler address SRA4 to the value it had before the call to printon.

P5. Additional Private MC's

Four private MC's are included. These are:

MC 1001 implements plot
MC 1002 implements printon
MC 1003 implements printoff
MC 1004 implements plotel

The private MC subroutine (USERMC) begins at address 3521H.
(See installation source code, attached.) Room has been left
to link two additional private MC's, and patching in more than
this is a simple matter. See section P6 for the allocation of
memory.

Space for additional private MC's is from 3605H through 367FH.
If more than this is needed, a reallocation of memory will be
required.

P6. Principal Memory Addresses

Allocations as delivered, (all hex):

<u>Origin</u>	<u>Furnished</u>	<u>Reallocated*</u>
BSTACK	3680	24K 3680
ESTACK	C900 = -3700	C900
BFUN	3700	3700
EFUN	C880 = -3780	C880
BVAR	3780	3780
EVAR	C280 = 3D80	C000
BPR	3D80	4000
EPR	A000 = -6000	9000
Highest address used	5FFF	6FFF
PPS starts at	3D89 3080	4009
PPS ends at	508E ✓	53DE
Dimension of PR in PPS	3000 decimal	7000
And is declared at	45FA	487A
And	461A	489A
An application program starts at	516E	? 5149? 53EE

*Write here in pencil any reallocations as per section P8.

These allocations are set for a 16K machine. For more or less memory, or to allocate space for private MC's follow the procedure given in Section P8.

P7. Utilities Provided

These are three stand-alone utilities provided, which are useful for various maintenance tasks. They are invoked from the front panel mode.

Note: Tiny-c is a copyrighted product. We license purchasers to make not more than five complete or partial copies for their own use, including printing, re-recording at higher speeds, and for backup, using TCMOD, DDUMP, PUTIL, or any other technique. You must place the tiny-c copyright notice on each copy made:

Copyright 1977
Tiny-c Associates

TCMOD (at 345AH) writes tiny-c to tape in the polyphase (P) mode with an auto-go record at the end. (Read the note in the box above for your responsibilities when copying parts of tiny-c).

In 3466H is stored the address of the highest memory byte that will be recorded. As furnished, it is the same as the "end of PPS" address given in Section P6, (or rounded to a higher value; it is alright to record too much.) If you have modified PPS, or reallocated memory, change this number to a sufficiently high address. Then do an SPJ345A. Start your recorder in the record mode, with the correct jack inserted for Polyphase recording. Then type G.

When the recording is complete, the HALT light will come on. Stop the recorder, and press cntl-Z to re-enter the front panel mode. The name of the recorded file is "TC". Load it by pushing reset and typing:

P
TC

It should auto-start, and PPS prompt should appear.
Note: Don't ever record anything on your original purchased copy of tiny-c. Save that tape as a backup.

DDUMP (at 3478H) is needed to make a separate copy of PPS, or to save an application program after a DONE message. (Read the note in the box above for your responsibilities when copying parts of tiny-c). If you ever change the allocation of memory, you will need to cold start tiny-c and reload PPS.

To do that you need a separate copy of PPS. To do this:

1. Load tiny-c
2. Enter the front panel mode.
3. Set HL to the PPS start address, and DE to its end address. See Section P6 above. You must get these addresses correct. Recording too much results in a copy unreadable by the .r command.
4. Set P to 3478H.
5. Start your recorder in the record mode, with the correct jack inserted for Polyphase recording.
6. When the recording is complete the HALT light will come on. Stop the recorder, and press cntl-Z to re-enter the front panel mode.

The name of the recorded file is "PPS", (not TC.PPS).

If you have done a lot of editing and the DONE message appears, then restarting tiny-c will erase your work. You can save the work using DDUMP:

1. Enter the front panel mode.
2. Set HL to the first byte used by an application. See Section P6.
3. Estimate how long the program being edited was when the DONE occurred. Convert to hex and add to the value in HL. This is your estimated "end of program" address. Examine memory there. Examine higher and lower addresses until you find the actual end of the program. The actual last byte is a carriage return (0D). The byte after that will be a null (00). The bytes before the 0D will be the text of the last program lines. Put into DE the address where you found the 0D byte.
4. Now follow steps 4, 5, and 6 above.

The name of the recorded file is "PPS", even though it is not the PPS program. Now you can hot start tiny-c with SPJ2006 and G commands. (Usually this works. Some crashes require reloading tiny-c.) When you get the PPS prompt (the single >), type:

.r PPS

which will load in the rescued program.

PUTIL (at 3492H) works like DDUMP. It is useful for printing a copy of PPS. It prints the hex address of the first character of each line. (Read the note in the box above for your responsibilities when copying parts of tiny-c.) Enter front panel mode, and set HL and DE as though you were going to DDUMP the PPS. Do an SPJ3492, turn your printer on and type G.

The USART is programmed by a CALL to the monitor 4.0 subroutine SETUP. This CALL is at 310CH. The four data bytes after this CALL, define the baud rate and the USART modes. The hex 16 at 310FH sets the baud at 300. Other choices are:

<u>Baud</u>	<u>Hex value at 310F</u>	<u>Baud</u>	<u>Hex value</u>
50	11	1200	19
75	12	1800	1A
1100	13	2400	1B
134.5	14	3600	1C
150	15	4800	1D
300	16	7200	1E
600	17	9600	1F
900	18		

P8. Reallocating Memory

There are two reasons to reallocate memory. One is because you have more than 16K bytes of memory. Another is to make room for private MC's. Follow these steps:

1. Calculate a new memory allocation. See Section 6.5.2 of the OWNER'S MANUAL. To help, a table of common allocations for the Poly 88 installation is given below.
2. Edit the new allocation addresses into 2036H through 2045H.
3. Do an SPJ2000 and a G. This is a cold start. No other start should be used for this step.
4. When you get the triple prompter, type:

~~.r PPS~~ .r TC.PPS

and load in the copy of PPS furnished with tiny-c, or your own version if you have modified it.

5. Type

.g

The PPS should start and its usual single prompter should appear.

6. Recalculate all the principal addresses of Section P6 above and document them in the space provided. Calculate the difference between the reallocated and furnished BPR. Add this to the PPS start and end addresses and the "dimension of PR at" addresses. Go into front panel mode, and examine memory to verify the address calculations are correct.
7. Edit the dimension of PR in the two places.
8. Now use TCMOD to record it all. Make sure the address stored in 3466H is high enough to cover the reallocated PPS.

In general, any time memory is reallocated, PPS must be reloaded, and a new TCMOD must be done. Useful addresses will change, and it is wise to document them for your own reference.

Here are some reasonable memory parameters for some popular memory amounts.

		Total memory: (round decimal number)				
	<u>Address</u>	<u>16K</u>	<u>20K</u>	<u>24K</u>	<u>32K</u>	<u>48K</u>
BSTACK	2036	3680	3680	3680	3700	3700
ESTACK	2038	C900	C900	C900	C800	C800
BFUN	203A	3700	3700	3700	3800	3800
EFUN	203C	C880	C880	C880	C700	C700
BVAR	203E	3780	3780	3780	3900	3900
EVAR	2040	C280	C000	BD80	B700	AD00
BPR	2042	3D80	4000	4280	4900	5300
EPR	2044	A000	9000	8000	6000	2000
End of memory is:		5FFF	6FFF	7FFF	9FFF	DFFF
PPS starts at:	3D80	3D89	4009	4289	4909	5309
PPS ends at:	53CF	508E	530E	558E	5C0E	660E
Dimension of PR is:						
(decimal)		3300	7000	10000	16800	32767
and is declared at:		45FA	487A	4AFA	517A	5B7A
and:		461A	489A	4B1A	519A	5B9A

An application

starts at: 516E 53EE 56EE 5CEE 66EE

So for 24K, do this for step 2: Enter the front panel mode and type:

L2036 J3680 L2038 JC900 L203A J3700....L2044 J8000

For step 7 do:

L4AFA 31 sp 30 sp 30 sp 30 sp 30

L4B1A 31 sp 30 sp 30 sp 30 sp 30

P9. File System Error Codes

When the .w or .r PPS commands are given, or when the file input/output functions are used, these error conditions may arise. The error letter is printed on the console.

- M Mode error. Only modes 'B' and 'P' are allowed. The bytes at these locations must be either 'B' (hex 42) or 'P' (hex 50):
345BH, 347BH, and 34DEH.
- T Type error. A record header with an unrecognized type (i.e. not 1,2,3,4 or 5) was read.
- H Checksum error in the header.
- D Checksum error in the data.
- C Attempting to read a comment record as data.

A record with the wrong filename in its header is ignored; nothing is printed.

P10. Reading Byte Mode Files

The Poly-88 installation is set up to do all input/output in the 2400 baud polyphase (P) mode. There is no B or P option in the .r or .w commands. If you need to read or write in the Byte mode, do this:

1. Enter the front panel mode.
2. Look at 34DEH. It should be hex 50. Change it to hex 42.
3. Type G to resume execution of tiny-c.

Be careful not to change any registers. When you have finished the Byte mode read or write, go back into front mode, and put a 50 back into 34DEH.

One time you will need to do this is when you read in your original purchased copy of PPS.

FILE 1

>AS 3100 610000

```

3100      0000      ;
3100      0010      SRA4      EQU      0C16H
3100      0020      SETUP     EQU      02ADH
3100      0030      IDRET      EQU      0064H
3100      0040      WH1        EQU      0C24H
3100      0050      WH0        EQU      0C20H
3100      0060      DSPLY      EQU      007FH
3100      0070      ;
3100      0080      ;
3100      0090      ;
3100      0100      ;POPEN -- NO ARGS; NO RETURNS. SETS UP USART FOR

3100      0110      ;          TI-735 OR SIMILAR PRINTER. MUST BE CALLE

3100      0120      ;          BEFORE USING PCHAR.
3100 2A 16 0C      0130      POPEN      LHL      SRA4
3103 22 19 31      0140      SHLD      OLD4      ;CAPTURE CURRENT SRA4 AD

3106 21 1B 31      0150      LXI        H,PISR
3109 22 16 0C      0160      SHLD      SRA4      ;SETUP OUR ISR ADDRESS
310C CD AD 02      0170      CALL      SETUP     ;SETUP USART
310F 16            0180      DB          016H      ;PORT 1, 300 BAUD
3110 AA            0190      DB          0AAH      ;ASSURES COMMAND MODE
3111 40            0200      DB          040H      ;RESET USART
3112 CE            0210      DB          0CEH      ;MODES FOR PRINTER
3113 00            0220      DB          0
3114 3E 21         0230      MVI        A,21H
3116 D3 01         0240      OUT         1          ;TURN ON USART
3118 C9            0250      RET
3119 01 00         0260      OLD4      DW          1
311B              0270      ;USART INTERRUPT HANDLER
311B 3A 47 31      0280      PISR      LDA      READY  ;TEST IF CHAR IS READY
311E B7            0290      ORA      A
311F CA 2D 31      0300      JZ          RUBOUT
3122 FE 55         0310      CPI        55H      ;TEST FOR RESTORE FLAG
3124 CA 38 31      0320      JZ          SHUTDOWN
3127 3A 48 31      0330      LDA      CHAR      ;GET BUFFERED CHAR
312A C3 2F 31      0340      JMP        CHOUT
312D 3E 7F         0350      RUBOUT     MVI        A,7FH  ;NO CHAR READY; SEND RUB

312F D3 00         0360      CHOUT      OUT         0      ;SEND IT
3131 AF            0370      XRA      A
3132 32 47 31      0380      STA      READY  ;ZERO CHAR READY FLAG
3135 C3 64 00      0390      JMP        IDRET
3138 AF            0400      SHUTDOWN    XRA      A
3139 32 47 31      0410      STA      READY
313C D3 01         0420      OUT         1          ;TURNS OFF USART
313E 2A 19 31      0430      LHL      OLD4      ;RESTORE INTRPT ADDR
3141 22 16 0C      0440      SHLD      SRA4
3144 C3 64 00      0450      JMP        IDRET
3147 00            0460      READY      DB          0      ;NONZERO IFF CHARACTER R

3148              0470      ;          ;IN CHAR.

```


3148 7F	0480	CHAR	DB	07FH	; CHARACTER BUFFER.
3149	0490	;			
3149	0500	; PCHAR -- CHARACTER IN A; NO RETURNS. OUTPUTS T			
3149	0510	; CHARACTER TO THE PRINTER. PERFORMS NEEDS			
3149	0520	; TRANSFORMATIONS TO CR AND HT.			
3149 F5	0530	PCHAR	PUSH	P	
314A FE 0D	0540		CPI	0DH	
314C CA 60 31	0550		JZ	CR	
314F FE 09	0560		CPI	09H	; TEST FOR HT
3151 CA 7A 31	0570		JZ	TAB	
3154 CD 8A 31	0580		CALL	BCHAR	; REGULAR CHARACTER; BUFF
3157 3A 89 31	0590		LDA	COLUMN	
315A 3C	0600		INR	A	
315B 32 89 31	0610	SETCOL	STA	COLUMN	; BUMP COLUMN COUNTER
315E F1	0620		POP	P	
315F C9	0630		RET		
3160 CD 8A 31	0640	CR	CALL	BCHAR	; SEND CR
3163 3E 0A	0650		MVI	A, 0AH	; SEND LF
3165 CD 8A 31	0660		CALL	BCHAR	
3168 3E 7F	0670		MVI	A, 7FH	; AND SOME SUBROUTS
316A CD 8A 31	0680		CALL	BCHAR	
316D CD 8A 31	0690		CALL	BCHAR	
3170 CD 8A 31	0700		CALL	BCHAR	
3173 CD 8A 31	0710		CALL	BCHAR	
3176 AF	0720		XRA	A	; ZERO COLUMN COUNTER
3177 C3 5B 31	0730		JMP	SETCOL	
317A 3E 20	0740	TAB	MVI	A, /	; SEND BLANKS
317C CD 49 31	0750		CALL	PCHAR	
317F 3A 89 31	0760		LDA	COLUMN	
3182 E6 07	0770		ANI	7H	
3184 C2 7A 31	0780		JNZ	TAB	; UNTIL COLUMN MOD 8 IS 0
3187 F1	0790		POP	P	
3188 C9	0800		RET		
3189 00	0810	COLUMN	DB	0	
318A F5	0820	BCHAR	PUSH	P	; ROUTINE TO BUFFER CHARA
318B 3A 47 31	0830	BLOOP	LDA	READY	
318E B7	0840		ORA	A	
318F C2 8B 31	0850		JNZ	BLOOP	; WAIT FOR LAST CHAR TO C
3192 F1	0860		POP	P	
3193 F5	0870		PUSH	P	
3194 32 48 31	0880		STA	CHAR	; BUFFER THIS CHAR
3197 AF	0890		XRA	A	
3198 2F	0900		CMA		
3199 32 47 31	0910		STA	READY	; SET READY FLAG
319C F1	0920		POP	P	; RESTORE A
319D C9	0930		RET		
319E	0940	;			
319E	0950	; RESTORE -- NO ARGS; NO RESULTS. SHUTS OF USART			
319E	0960	; RESTORES OLD SRA4.			
319E AF	0970	RESTORE	XRA	A	
319F CD 8A 31	0980		CALL	BCHAR	; FLUSH BUFFER AND USART
31A2 CD 8A 31	0990		CALL	BCHAR	
31A5 CD 8A 31	1000		CALL	BCHAR	
31A8 3E 55	1010		MVI	A, 55H	; SPECIAL READY FLAG SIGN
31AA 32 47 31	1020		STA	READY	; PISR TO SHUT DOWN USA
31AD 3A 47 31	1030	RLOOP	LDA	READY	; Now delay til shutdown
31B0 B7	1040		ORA	A	
31B1 C2 AD 31	1050		JNZ	RLOOP	
31B4 C9	1060		RET		


```

1070 ;
1080 ; Hooks up printer to WH1.
1090 HOOKUP CALL POPEN
1100 LHL D WH1+1
1110 SHLD OLDWH ;SAVE OLD WH1 ADDRESS
1120 LXI H,NEWWH
1130 SHLD WH1+1
1140 RET
1150 NEWWH PUSH P ;CALLS TO WH1 COME HERE
1160 CALL DSPLY
1170 CALL PCHAR
1180 POP P
1190 RET
1200 ;
1210 ;UNHOOK -- MAIN PROGRAM. UNHOOKS PRINTER FROM WH1

1220 ; WARNING - DONT USE UNLESS HOOKUP WAS RUN
E
1230 UNHOOK CALL RESTORE ;RESTORES SRA4 AND USART
1240 LHL D OLDWH
1250 SHLD WH1+1
1260 RET
1270 OLDWH DW 1
>TF

```

FILE 2

>ASS 31DA 61DA

```

31DA 0000 ; *****
31DA 0010 ; **** FILE SYSTEM ****
31DA 0020 ; *****
31DA 0030 ;
31DA 0040 ;BY TOM GIBSON, TINY-C ASSOCIATES, BOX 269,
31DA 0050 ; HOLMDEL, N. J., 07733
31DA 0060 ; JULY 15, 1977.
31DA 0070 ;
31DA 0080 ;A COMPLETE SUBROUTINE PACKAGE FOR MANIPULATING
31DA 0090 ; FIVE RECORD TYPES OF A POLY-8. SUBROUTINES AR
31DA 0100 ; FURNISHED TO OPEN A FILE, AND READ OR WRITE A
31DA 0110 ; THE FIVE RECORD TYPES. THE ROUTINES OPEN, WRA
31DA 0120 ; WRCOM, WREOF, WRAGO, WRDATA, AND READ ARE DEF
31DA 0130 ; IN THE LISTING BELOW.
31DA 0140 ;
31DA 0150 ;OPENS A FILE. THE LETTER 'B' OR 'P' IS IN A, FO
31DA 0160 ; OR POLYPHASE RECORDING MODE. (HL) IS THE ADDR
31DA 0170 ; A CHARACTER STRING, THE FILE NAME. IF SHORTER
31DA 0180 ; 8 BYTES, ITS END IS SIGNALLED WITH A NULL BYTE
31DA 0190 OPEN EQU $
31DA F5 0200 PUSH P
31DB 11 4B 33 0210 LXI D,FNAME ;FILENAME -> FNAME
31DE 01 F8 FF 0220 LXI B,-8
31E1 CD 00 01 0230 CALL MOVE

```


31E4 21 4B 33	0240	LXI	H,FNAME	;LOOK FOR NULL IN FNAME
31E7 06 08	0250	MVI	B,8	
31E9 CD 56 32	0260	CALL	NLSCAN	
31EC 13	0270	INX	D	;POINTS TO NULL, IF ANY,
31ED 21 52 33	0280	LXI	H,FNAME+7	;BEYOND FNAME.
31F0 06 20	0290	MVI	B,' '	
31F2 CD 50 34	0300	CALL	BZAP	;BLANKS FNAME FROM NULL
31F5 11 65 33	0310	LXI	D,WNAME	;NOW MAKE A COPY IN WNAME
31F8 21 4B 33	0320	LXI	H,FNAME	
31FB 01 F8 FF	0330	LXI	B,-8	
31FE CD 00 01	0340	CALL	MOVE	
3201 21 00 00	0350	LXI	H,0	
3204 22 6D 33	0360	SHLD	WRN	
3207 F1	0370	POP	P	
3208 FE 42	0380	CPI	'B'	;CHECK FOR BYTE OR POLY
320A CA 16 32	0390	JZ	BYTE	
320D FE 50	0400	CPI	'P'	
320F CA 20 32	0410	JZ	POLY	
3212 3E 4D	0420	MVI	A,'M'	;MODE ERROR
3214 B7	0430	ORA	A	
3215 C9	0440	RET		
3216 CD AD 02	0450	CALL	SETUP	
3219 06	0460	DB	006H	;DEVICE 0, 300 BAUD
321A AA	0470	DB	0AAH	;ASSURES COMMAND MODE IN
321B 40	0480	DB	040H	;RESET USART
321C CE	0490	DB	0CEH	;MODES FOR BYTE STANDARD
321D 00	0500	DB	0	;LEAVE USART IDLING
321E AF	0510	XRA	A	
321F C9	0520	RET		;RETURN 0 MEANS O.K.
3220 CD AD 02	0530	CALL	SETUP	
3223 05	0540	DB	005H	;DEVICE 0, 2400 BAUD
3224 AA	0550	DB	0AAH	
3225 40	0560	DB	040H	
3226 0C	0570	DB	00CH	;MODES FOR POLYPHASE
3227 E6	0580	DB	0E6H	;TWO SYNC BYTES
3228 E6	0590	DB	0E6H	
3229 00	0600	DB	0	
322A AF	0610	XRA	A	
322B C9	0620	RET		
322C	0630			
322C	0640			;WRITES AN ABSOLUTE BINARY FILE FROM (HL) TO (DE)
322C	0650	WRABB EQU	\$	
322C 22 70 33	0660	SHLD	WADR	;SET UP HEADER INFO
322F AF	0670	XRA	A	
3230 32 72 33	0680	STA	WTYPE	;ABS BINARY IS TYPE 0
3233 CD 39 32	0690	CALL	WLEN	;COMPUTES FILE LENGTH
3236 C3 92 32	0700	JMP	DUMPPR	;THIS DOES ALL THE WORK,
3239	0710			RETURNS TO CALLER.
3239	0720			
3239	0730			;COMPUTES 1 + (DE) - (HL) AND PUTS IN LENGTH
3239 13	0740	WLEN INX	D	
323A 7B	0750	MOV	A,E	
323B 95	0760	SUB	L	
323C 32 63 33	0770	STA	LENGTH	
323F 7A	0780	MOV	A,D	
3240 9C	0790	SBB	H	
3241 32 64 33	0800	STA	LENGTH+1	
3244 C9	0810	RET		
3245	0820			

3245		0830	;WRITES A COMMENT RECORD; TEXT FROM (HL) TO NULL		
3245		0840	WRCOM	EQU	\$
3245 22 70 33		0850		SHLD	WADR ;SETUP HEADER INFO
3248 3E 01		0860		MVI	A,1
324A 32 72 33		0870		STA	WTYPE ;COMMENT RECORD IS TYPE
324D CD 56 32		0880		CALL	NLSCAN ;FINDS LAST BYTE
3250 CD 39 32		0890		CALL	WRLN ;COMPUTES RECORD'S LENGT
3253 C3 92 32		0900		JMP	DUMPPR
3256		0910	;		
3256		0920	;SCANS FOR NULL BYTE; FROM (HL) THRU (HL)+(B)-1.		
3256		0930	; IF A NULL IS FOUND; DE POINTS TO BYTE BEFORE		
3256		0940	; NULL; AND (B) IS THE NUMBER OF BYTES NOT EXAM		
3256		0950	; IF A NULL IS NOT FOUND; (DE) IS THE LAST BYTE		
3256		0960	; EXAMINED; (B) IS ZERO. IN EITHER CASE; HL IS		
3256		0970	; UNCHANGED.		
3256		0980	NLSCAN	EQU	\$
3256 54		0990		MOV	D,H ;COPY (HL) INTO DE
3257 5D		1000		MOV	E,L
3258 1A		1010	SCLOOP	LDAX	D
3259 B7		1020		ORA	A
325A CA 63 32		1030		JZ	NULL
325D 05		1040		DCR	B
325E C8		1050		RZ	;NO NULL FOUND
325F 13		1060		INX	D
3260 C3 58 32		1070		JMP	SCLOOP
3263 1B		1080	NULL	DCX	D
3264 C9		1090		RET	
3265		1100	;		
3265		1110	;WRITES AN EOF RECORD		
3265		1120	WEOF	EQU	\$
3265 21 01 00		1130		LXI	H,1 ;WRITE LENGTH IS 1
3268 22 63 33		1140		SHLD	LENGTH
326B 3E 02		1150		MVI	A,2 ;TYPE IS 2
326D 32 72 33		1160		STA	WTYPE
3270 C3 92 32		1170		JMP	DUMPPR
3273		1180	;		
3273		1190	;WRITES AN AUTO-GO RECORD; USING THE ADDRESS (HL)		
3273		1200	WRAGO	EQU	\$
3273 22 70 33		1210		SHLD	WADR
3276 21 01 00		1220		LXI	H,1 ;LENGTH IS 1
3279 22 63 33		1230		SHLD	LENGTH
327C 3E 03		1240		MVI	A,3 ;TYPE IS 3
327E 32 72 33		1250		STA	WTYPE
3281 C3 92 32		1260		JMP	DUMPPR
3284		1270	;		
3284		1280	;WRITES A SET OF DATA RECORDS READABLE INTO ANY		
3284		1290	; DATA IS FROM (HL) THRU (DE).		
3284		1300	WRDATA	EQU	\$
3284 22 70 33		1310		SHLD	WADR
3287 3E 04		1320		MVI	A,4
3289 32 72 33		1330		STA	WTYPE ;TYPE IS 4
328C CD 39 32		1340		CALL	WRLN ;COMPUTES FILE LENGTH.
328F C3 92 32		1350		JMP	DUMPPR
3292		1360	;		
3292		1370	;WRITES A SET OF DATA RECORDS AS DESCRIBED IN HE		
3292 2A 16 0C		1380	DUMPPR	LHLD	SRA4 ;SET UP INTERRUPT ADDRES

3295	22 19 31	1390		SHLD	OLD4	
3298	21 3F 33	1400		LXI	H,TISR	
329B	22 16 0C	1410		SHLD	SRA4	
329E	21 64 33	1420	DNEXT	LXI	H,LENGTH+1	;HI BYTE IS THE
32A1	7E	1430		MOV	A,M	; OF FULL SIZE
32A2	B7	1440		ORA	A	
32A3	CA B8 32	1450		JZ	LAST	
32A6	35	1460		DCR	M	
32A7	AF	1470		XRA	A	
32A8	32 6F 33	1480		STA	WLEN	;ZERO LENGTH MEANS FULL
32AB	CD CC 32	1490		CALL	DUMP	; RECORD (256 BYTES)
32AE	2A 70 33	1500		LHLD	WADR	;WRITE RECORD, THEN BUMP
32B1	24	1510		INR	H	; BY 256.
32B2	22 70 33	1520		SHLD	WADR	
32B5	C3 9E 32	1530		JMP	DNEXT	;GO BACK FOR ANOTHER REC
32B8	2B	1540	LAST	DCX	H	;POINTS TO LD BYTE OF LE
32B9	7E	1550		MOV	A,M	; TEST IF ANY BYTES LEF
32BA	B7	1560		ORA	A	
32BB	CA C5 32	1570		JZ	DONE	;NONE, SO ALL DONE.
32BE	32 6F 33	1580		STA	WLEN	;WRITE ONE PARTIAL RECOR
32C1	CD CC 32	1590		CALL	DUMP	; (1-255 BYTES), THEN D
32C4	AF	1600		XRA	A	;RETURN A 0
32C5	2A 19 31	1610	DONE	LHLD	OLD4	;RESTORE OLD INTERRUPT A
32C8	22 16 0C	1620		SHLD	SRA4	
32CB	C9	1630		RET		
32CC		1640				
32CC		1650				;WRITES ONE RECORD, INCLUDING INTERRECORD GAP, H
32CC		1660				; CHECKSUM ON HEADER, DATA, AND CHECKSUM ON DAT
32CC		1670				; DISPLAYS ADDRESS OF RECORD ABOUT TO BE WRITTE
32CC		1680				; ALSO STARTS AND STOPS MOTOR AND USART.
32CC	3E 21	1690	DUMP	MVI	A,021H	
32CE	D3 01	1700		OUT	1	;TURN ON USART AND MOTOR
32D0	2A 70 33	1710		LHLD	WADR	
32D3	EB	1720		XCHG		
32D4	CD D1 03	1730		CALL	DEOUT	;DISPLAY ADDRESS
32D7	CD 8D 03	1740		CALL	CRLF	
32DA	21 FF 8F	1750		LXI	H,08FFFH	
32DD	2B	1760	DELAY	DCX	H	;DELAY ~ 0.25 SECS.
32DE	7C	1770		MOV	A,H	
32DF	B7	1780		ORA	A	
32E0	C2 DD 32	1790		JNZ	DELAY	
32E3	0E 40	1800		MVI	C,64	;WRITE 64 SYNCs
32E5	3E E6	1810		MVI	A,0E6H	
32E7	CD 2E 33	1820	D2	CALL	T0	
32EA	0D	1830		DCR	C	
32EB	C2 E7 32	1840		JNZ	D2	
32EE	3E 01	1850		MVI	A,001H	;WRITE SOH
32F0	CD 2E 33	1860		CALL	T0	
32F3	3E 0E	1870		MVI	A,14	;WRITE 14 BYTE HEADER

32F5 21 65 33	1880	LXI	H, WNAME	
32F8 CD 18 33	1890	CALL	PUT	
32FB 3A 6F 33	1900	LDA	WLEN	;WRITE DATA
32FE 2A 70 33	1910	LHLD	WADR	
3301 CD 18 33	1920	CALL	PUT	
3304 2A 6D 33	1930	LHLD	WRN	
3307 23	1940	INX	H	
3308 22 6D 33	1950	SHLD	WRN	
330B AF	1960	XRA	A	;PUSH OUT LAST BYTES IN
330C CD 2E 33	1970	CALL	TO	; AND USART.
330F CD 2E 33	1980	CALL	TO	
3312 CD 2E 33	1990	CALL	TO	
3315 D3 01	2000	OUT	1	;TURN OFF USART AND MOTO
3317 C9	2010	RET		
3318	2020			
3318	2030			;OUTPUTS AN ARRAY OF BYTES FROM (HL) TO (HL)+(A)
3318 06 00	2040	PUT	MVI B,0	; FOLLOWED BY THEIR CHE
331A 4F	2050		MOV C,A	;LENGTH -> C
331B 7E	2060	P2	MOV A,M	
331C 23	2070		INX H	
331D F5	2080		PUSH P	
331E 80	2090		ADD B	
331F 47	2100		MOV B,A	
3320 F1	2110		POP P	
3321 CD 2E 33	2120		CALL TO	
3324 0D	2130		DCR C	
3325 C2 1B 33	2140		JNZ P2	
3328 78	2150		MOV A,B	;CHECKSUM -> A
3329 2F	2160		CMA	
332A 3C	2170		INR A	
332B C3 2E 33	2180		JMP TO	;WRITE -CHECKSUM AND RET
332E	2190			
332E	2200			;WRITES (A). RESTORES ALL REGISTERS.
332E E5	2210	TO	PUSH H	
332F 21 08 0C	2220		LXI H,TBUFF	
3332 F5	2230		PUSH P	
3333 7E	2240	TO2	MOV A,M	;WAIT FOR LAST CHAR TO C
3334 B7	2250		DRA A	
3335 C2 33 33	2260		JNZ TO2	
3338 23	2270		INX H	;POINTS TO CHAR BUFFER
3339 F1	2280		POP P	
333A 77	2290		MOV M,A	;CHAR -> BUFFER
333B 2B	2300		DCX H	;POINTS TO FLAG
333C 34	2310		INR M	;SETS FLAG FOR TISR
333D E1	2320		POP H	;ALL REGS NOW RESTORED
333E C9	2330		RET	
333F	2340			
333F	2350			;USART INTERRUPT HANDLER -- TRANSMITS THE CHARAC
333F	2360			; TBUFF+1. IT DOES NOT CHECK THE FLAG. THEREFOR
333F	2370			; PROGRAM FILLING TBUFF MUST BE FASTER THAN THE
333F	2380			; OR ELSE IT MUST TURN IT OFF. THE DUMP PROGRAM
333F	2390			; IS FAST ENOUGH, AND OUTSIDE OF DUMP THE USART
333F AF	2400	TISR	XRA A	

3340 32 08 0C	2410	STA	TBUFF	;FLAG OFF, FOR TO.
3343 3A 09 0C	2420	LDA	TBUFF+1	
3346 D3 00	2430	OUT	0	;THERE IT GOOOOOOOES!!!!
3348 C3 64 00	2440	JMP	IQRET	
334B	2450	;		
334B	2460	;DATA AREAS		
334B	2470	FNAME DS	8	
3353	2480	RNAME DS	8	
335B	2490	RRN DS	2	
335D	2500	RLEN DS	1	
335E	2510	RADR DS	2	
3360	2520	RTYPE DS	1	
3361	2530	DADR DS	2	
3363	2540	;		
3363	2550	LENGTH DS	2	
3365	2560	WNAME DS	8	
336D	2570	WRN DS	2	
336F	2580	WLEN DS	1	
3370	2590	WADR DS	2	
3372	2600	WTYPE DS	1	
3373	2610	;		
3373	2620	GET EQU	029AH	
3373	2630	MOVE EQU	0100H	
3373	2640	WH2 EQU	0C28H	
3373	2650	TBUFF EQU	0C08H	
3373	2660	DEOUT EQU	03D1H	
3373	2670	CRLF EQU	038DH	
>TF				

FILE 3

>ASS 3373 6373

3373	0000	;		
3373	0010	;READS ONE RECORD OF ANY TYPE. STARTS AND STOPS		
3373	0020	; AND USART, SO AN ARBITRARY AMOUNT OF PROCESSI		
3373	0030	; BE DONE BETWEEN READS. (HL) MUST BE GIVEN AS		
3373	0040	; ADDRESS OF A DATA AREA, JUST IN CASE RECORD I		
3373	0050	; DATA RECORD. RETURNS 0 IN A FOR NO ERROR, QTH		
3373	0060	; A ONE CHARACTER SEMI-MNEMONIC INDICATOR OF TH		
3373	0070	; ERROR.		
3373	0080	;IF RECORD IS A DATA RECORD, HL IS BUMPED BY THE		
3373	0090	; LENGTH OF THE RECORD. IF THE RECORD IS EITHER		
3373	0100	; A DATA OR ABSOLUTE RECORD, THE RECORD NUMBER		
3373	0110	; RETURNED IN DE.		
3373	0120	READ EQU	\$	
3373 22 61 33	0130	SHLD DADR		;SAVE DATA ADDRESS
3376 2A 16 0C	0140	LHLD SRA4		;SETUP INTERRUPT ADDRESS
3379 22 19 31	0150	SHLD OLD4		
337C 21 54 00	0160	LXI H,0054H		;MON. 4.0 HANDLER
337F 22 16 0C	0170	SHLD SRA4		
3382 CD 11 34	0180	R2 CALL	HEADER	;SCAN FOR AND READ A HEA

3385 FA A9 33	0190	JM	HERR	
3388 C2 82 33	0200	JNZ	R2	; NO MATCH; GET NEXT HEAD
338B 3A 60 33	0210	LDA	RTYPE	; BRANCH 5 WAYS ON RECORD
338E B7	0220	ORA	A	
338F CA BD 33	0230	JZ	RABB	
3392 3D	0240	DCR	A	
3393 CA D5 33	0250	JZ	RCMT	
3396 3D	0260	DCR	A	
3397 CA EA 33	0270	JZ	REDF	
339A 3D	0280	DCR	A	
339B CA F1 33	0290	JZ	RAGO	
339E 3D	0300	DCR	A	
339F CA F8 33	0310	JZ	RDATA	
33A2 CD B0 33	0320	CALL	MOFF	; TYPE ERROR; RETURN 'T'
33A5 3E 54	0330	MVI	A, 'T'	
33A7 B7	0340	ORA	A	
33A8 C9	0350	RET		
33A9 CD B0 33	0360	HERR CALL	MOFF	; HEADER CHECKSUM ERROR
33AC 3E 48	0370	MVI	A, 'H'	
33AE B7	0380	ORA	A	
33AF C9	0390	RET		
33B0 AF	0400	MOFF XRA	A	; TURN OFF MOTOR AND USAR
33B1 D3 01	0410	OUT	1	
33B3 2A 19 31	0420	LHLD	OLD4	
33B6 22 16 0C	0430	SHLD	SRA4	
33B9 2A 61 33	0440	LHLD	DADR	; RETURN DATA ADDRESS (IF
33BC C9	0450	RET		
33BD 2A 5E 33	0460	RABB LHLD	RADR	; READ ABSOLUTE BINARY
33C0 3A 5D 33	0470	RSTUFF LDA	RLEN	
33C3 4F	0480	MOV	C, A	
33C4 CD 9A 02	0490	CALL	GET	
33C7 2A 5B 33	0500	LHLD	RRN	; RETURN REC NO IN DE
33CA EB	0510	XCHG		
33CB CA B0 33	0520	JZ	MOFF	; RETURNS A 0 FOR O.K.
33CE CD B0 33	0530	CALL	MOFF	; DATA ERROR; RETURN A 'D'
33D1 3E 44	0540	MVI	A, 'D'	
33D3 B7	0550	ORA	A	
33D4 C9	0560	RET		
33D5 3A 5D 33	0570	RCMT LDA	RLEN	
33D8 4F	0580	MOV	C, A	; LENGTH -> C
33D9 CD 28 0C	0590	RC2 CALL	WH2	; GET COMMENT CHARACTER
33DC CD 7F 00	0600	CALL	DSPLY	
33DF 0D	0610	DCR	C	
33E0 C2 D9 33	0620	JNZ	RC2	
33E3 CD B0 33	0630	CALL	MOFF	
33E6 3E 43	0640	MVI	A, 'C'	; NOT AN ERROR PER SE, BU
33E8 B7	0650	ORA	A	; CALLING PROGRAM.
33E9 C9	0660	RET		
33EA CD B0 33	0670	REDF CALL	MOFF	
33ED 3E 45	0680	MVI	A, 'E'	; ADVISE CALLING PROGRAM
33EF B7	0690	ORA	A	; OF END OF FILE
33F0 C9	0700	RET		
33F1 CD B0 33	0710	RAGO CALL	MOFF	
33F4 2A 5E 33	0720	LHLD	RADR	; JUMP TO THE ADDRESS REA
33F7 E9	0730	PCHL		
33F8 2A 61 33	0740	RDATA LHLD	DADR	

33FB CD C0 33	0750	CALL	RSTUFF	
33FE C0	0760	RNZ		; RETURN IF ERROR
33FF 3A 5D 33	0770	LDA	RLEN	; BUMP DATA ADDRESS BY RL
3402 2A 61 33	0780	LHLD	DADR	; RETURN ORIGINAL (HL)+RE
3405 B7	0790	ORA	A	
3406 CA 0E 34	0800	JZ	RD2	; RLEN=256
3409 85	0810	ADD	L	; ADD RLEN (< 256)
340A 6F	0820	MOV	L,A	
340B D2 0F 34	0830	JNC	R3	
340E 24	0840	RD2	INR	H ; ADD 256, OR PERFORM CAR
340F AF	0850	R3	XRA	A ; RETURN 0, FOR NO ERROR
3410 C9	0860		RET	
3411	0870			
3411	0880			; STARTS MOTOR; STARTS USART IN READ MODE; READS
3411	0890			; HEADER INTO RNAME;... MATCHES RNAME WITH FNAME
3411	0900			; RETURNS 0 ON MATCH; NOT 0 & PLUS ON NOT MATCH
3411	0910			; MINUS ON ERROR. RETURNS WITH MOTOR RUNNING!!!
3411	0920	HEADER	EQU	\$
3411 3E 96	0930		MVI	A,96H
3413 D3 01	0940		OUT	1 ; MOTOR AND USART ON
3415 CD 28 0C	0950		CALL	WH2
3418 FE E6	0960	SYNC	CPI	0E6H
341A C2 11 34	0970		JNZ	HEADER
341D CD 28 0C	0980		CALL	WH2
3420 FE 01	0990		CPI	1
3422 C2 18 34	1000		JNZ	SYNC
3425 21 53 33	1010		LXI	H,RNAME
3428 0E 0E	1020		MVI	C,14
342A CD 9A 02	1030		CALL	GET
342D C2 40 34	1040		JNZ	CSERR
3430 21 53 33	1050		LXI	H,RNAME
3433 11 4B 33	1060		LXI	D,FNAME
3436 0E 08	1070		MVI	C,8
3438 CD 44 34	1080		CALL	COMPARE
343B C8	1090		RZ	
343C 3E 01	1100		MVI	A,1
343E B7	1110		ORA	A
343F C9	1120		RET	
3440 3E FF	1130	CSERR	MVI	A,-1
3442 B7	1140		ORA	A
3443 C9	1150		RET	
3444	1160			
3444	1170			; COMPARES (C) BYTES FROM (DE) AND (HL). RETURNS
3444	1180			; MATCH; NOT ZERO ON NO MATCH.
3444 1A	1190	COMPARE	LDAX	D
3445 BE	1200		CMP	M
3446 C0	1210		RNZ	
3447 0D	1220		DCR	C
3448 C8	1230		RZ	
3449 23	1240		INX	H
344A 13	1250		INX	D
344B C3 44 34	1260		JMP	COMPARE
344E	1270			
344E	1280			; STORE 0'S FROM (DE) THRU (HL)
344E 06 00	1290	ZERO	MVI	B,0
3450 7D	1300	BZAP	MOV	A,L
3451 93	1310		SUB	E

3452 7C	1320	MOV	A,H
3453 9A	1330	SBB	D
3454 D8	1340	RC	
3455 70	1350	MOV	M,B
3456 2B	1360	DCX	H
3457 C3 50 34	1370	JMP	BZAP
>TF			

FILE 4

>ASS 345A 645A

345A	0000	;	
345A	0010	;TINY-C MODULE DUMPER.	
345A 3E 50	0020	TCMOD	MVI A,'P'
345C 21 75 34	0030		LXI H,TCNAME
345F CD DA 31	0040		CALL OPEN
3462 21 00 20	0050		LXI H,2000H
3465 11 00 51	0060		LXI D,5100H ;ENOUGH TO COVER TC + PA
3468	0070	; + INSTALLATION (EVEN	
3468	0080	; + TC.SYS	
3468 CD 2C 32	0090	CALL	WRABB
346B	0100	;	
346B 21 06 20	0110	LXI	H,2006H
346E CD 73 32	0120	CALL	WRAGD ;HOT START
3471 76	0130	HALT	HLT
3472 C3 71 34	0140	JMP	HALT
3475 74	0150	TCNAME	DB 'T'
3476 63	0160		DB 'C'
3477 00	0170		DB 0
3478	0180	;	
3478	0190	;DATA DUMPER. USER MUST SET DE AND HL. THE AREA	
3478	0200	; (HL) THRU (DE) IS WRITTEN AS A DATA FILE, THE	
3478	0210	; THE NAME OF THE FILE IS 'DATA'.	
3478 D5	0220	DDUMP	PUSH D
3479 E5	0230		PUSH H
347A 3E 50	0240		MVI A,'P'
347C 21 8D 34	0250		LXI H,DNAME
347F CD DA 31	0260		CALL OPEN
3482 E1	0270		POP H
3483 D1	0280		POP D
3484 CD 84 32	0290		CALL WRDATA
3487 CD 65 32	0300		CALL WREOF
348A C3 71 34	0310		JMP HALT
348D 64	0320	DNAME	DB 'D'
348E 61	0330		DB 'A'
348F 74	0340		DB 'T'
3490 61	0350		DB 'A'
3491 00	0360		DB 0
3492	0370	;	
3492	0380	;PRINT UTILITY. USER MUST SET HL AND DE. PRINTS	
3492	0390	;(HL)..(DE) INCLUSIVE. PUTS HEX ADDRESS AT BEGIN	
3492	0400	;OF EACH LINE.	
3492 E5	0410	PUTIL	PUSH H
3493 D5	0420		PUSH D

3494	CD	B5	31	0430	CALL	HOOKUP	
3497	D1			0440	POP	D	
3498	E1			0450	POP	H	
3499	7E			0460	PLOOP	MOV	A,M
349A	B7			0470	ORA	A	
349B	C2	A0	34	0480	JNZ	\$+2	;NULLS PRINT AS "
349E	3E	22		0490	MVI	A,'"/	
34A0	CD	24	0C	0500	CALL	WH1	
34A3	23			0510	INX	H	
34A4	FE	0D		0520	CPI	0DH	
34A6	C2	B9	34	0530	JNZ	NEXT	
34A9	7C			0540	MOV	A,H	;PRINT (HL) IN HEX
34AA	CD	C9	34	0550	CALL	AOUT	
34AD	7D			0560	MOV	A,L	
34AE	CD	C9	34	0570	CALL	AOUT	
34B1	3E	20		0580	MVI	A,'"/	
34B3	CD	24	0C	0590	CALL	WH1	
34B6	CD	24	0C	0600	CALL	WH1	
34B9	7D			0610	NEXT	MOV	A,L
34BA	93			0620	SUB	E	
34BB	C2	99	34	0630	JNZ	PLOOP	
34BE	7C			0640	MOV	A,H	
34BF	9A			0650	SBB	D	
34C0	C2	99	34	0660	JNZ	PLOOP	
34C3	CD	CE	31	0670	CALL	UNHOOK	
34C6	C3	71	34	0680	JMP	HALT	
34C9	F5			0690	AOUT	PUSH	P
34CA	0F			0700	RRC		; (A) -> PRINTER IN HEX
34CB	0F			0710	RRC		
34CC	0F			0720	RRC		
34CD	0F			0730	RRC		
34CE	CD	D2	34	0740	CALL	NYBBLE	
34D1	F1			0750	POP	P	
34D2	E6	0F		0760	NYBBLE	ANI	15
34D4	C6	90		0770	ADI	90H	
34D6	27			0780	DAA		
34D7	CE	40		0790	ACI	40H	
34D9	27			0800	DAA		
34DA	C3	24	0C	0810	JMP	WH1	
>TF							

FILE 5

>ASS 34DD 64DD

34DD	0000	;THREE SUPPORTED ENTRY POINTS	
34DD	0010	MCESET	EQU 202BH
34DD	0020	TOPTOI	EQU 202EH
34DD	0030	PUSHK	EQU 2031H
34DD	0040	;THREE UNSUPPORTED ENTRY POINTS INTO TINY C. THE	
L			
34DD	0050	; HAVE TO BE CHANGED FOR NEW RELEASES.	
34DD	0060	DENEG	EQU 2174H
34DD	0070	DREM	EQU 2131H
34DD	0080	PZERO	EQU 21C5H
34DD	0090	;ONE MON 4.0 REFERENCE, (PLUS THE ONES DEFINED A	
34DD	0100	KBUFF	EQU 0C0CH
34DD	0110	;	
34DD	0120	;POLY-88 MONITOR 4.0 INSTALLATION OF TINY-C.	

34DD		0130	; THIS PROGRAM IS COPYRIGHTED:	
34DD		0140	;COPYRIGHT, 1977, T. A. GIBSON	
34DD		0150	;	
34DD 3E 50		0160	FOPEN	MVI A,'P' ;ALL FILES WILL BE POLYP
34DF CD DA 31		0170	CALL	OPEN
34E2 AF		0180	XRA	A ;CANT HAVE AN ERROR
34E3 C9		0190	RET	
34E4 E5		0200	FREAD	PUSH H ;CANT CLOBBER HL
34E5 CD 73 33		0210	CALL	READ
34E8 C2 FD 34		0220	JNZ	FREND ;END OR ERROR
34EB CD D1 03		0230	CALL	DEOUT ;STROKE THE USER
34EE 3E 20		0240	MVI	A,' '
34F0 CD 24 0C		0250	CALL	WH1
34F3 D1		0260	POP	D ;CALCULATE BYTES READ ->
34F4 D5		0270	PUSH	D
34F5 CD 74 21		0280	CALL	DENEG
34F8 19		0290	DAD	D
34F9 EB		0300	XCHG	
34FA E1		0310	POP	H ;RESTORE ORIGINAL HL
34FB AF		0320	XRA	A ;NO ERROR
34FC C9		0330	RET	
34FD E1		0340	FREND	POP H ;RESTORE ORIGINAL HL
34FE FE 45		0350	CPI	'E' ;END OR ERROR?
3500 C2 0C 35		0360	JNZ	FRERR
3503 3E 0D		0370	MVI	A,0DH ;END, STROKE USER
3505 CD 24 0C		0380	CALL	WH1
3508 3E FF		0390	MVI	A,-1 ;END SIGNAL FOR TINY-C.
350A B7		0400	DRA	A
350B C9		0410	RET	
350C CD 24 0C		0420	FRERR	CALL WH1 ;ERROR;
350F B7		0430	DRA	A ;SIGNAL TINY-C
3510 C9		0440	RET	
3511 3A 0C 0C		0450	CHRDY	LDA KBUFF ;FLAG, 0 MEANS READY
3514 B7		0460	DRA	A
3515 CA 1A 35		0470	JZ	RDY
3518 AF		0480	XRA	A ;RETURN 0 WHEN NOT READY
3519 C9		0490	RET	
351A 3A 0D 0C		0500	RDY	LDA KBUFF+1 ;COPY OF THE CHARACTER
351D B7		0510	DRA	A
351E C0		0520	RNZ	
351F 3C		0530	INR	A ;NULL BYTE RETURNED AS S
3520 C9		0540	RET	
3521		0550	;	
3521 7D		0560	USERMC	MOV A,L ;1000 ALREADY SUBTRACTED
3522 FE 01		0570	CPI	1
3524 CA 43 35		0580	JZ	MC1001
3527 FE 02		0590	CPI	2
3529 CA B9 35		0600	JZ	MC1002
352C FE 03		0610	CPI	3
352E CA BF 35		0620	JZ	MC1003
3531 FE 04		0630	CPI	4
3533 CA E6 35		0640	JZ	MC1004
3536 FE 05		0650	CPI	5
3538 CA 2B 20		0660	JZ	MCESET ;TWO SPARES
353B FE 06		0670	CPI	6
353D CA 2B 20		0680	JZ	MCESET
3540 C3 2B 20		0690	JMP	MCESET
3543		0700	;PLOT MC, ARGS ARE ROW COLUMN ONOFF. RETURNS 0 0	
3543		0710	; 1 IF OFF SCREEN.	
3543 CD 2E 20		0720	MC1001	CALL TOPTOI

3546 D5	0730	PUSH	D	
3547 CD 2E 20	0740	CALL	TOPTOI	
354A D5	0750	PUSH	D	
354B CD 2E 20	0760	CALL	TOPTOI	
354E 43	0770	MOV	B,E	;ROW
354F D1	0780	POP	D	
3550 4B	0790	MOV	C,E	;COLUMN
3551 D1	0800	POP	D	
3552 7B	0810	MOV	A,E	;ON/OFF
3553 CD 5C 35	0820	CALL	PLOT	
3556 5F	0830	MOV	E,A	;RESULTS -> DE
3557 AF	0840	XRA	A	
3558 57	0850	MOV	D,A	
3559 C3 31 20	0860	JMP	PUSHK	
355C	0870			
355C	0880			; DRIVER FOR POLYMORPHIC VTI. ROW (0-47) IN B. CO
355C	0890			; (0-127) IN C. ONOFF IN A; ZERO MEANS TURN SPD
355C	0900			; NONZERO MEANS ON. USES ALL REGISTERS, AND DOE
355C	0910			; RESTORE THEM.
355C F5	0920	PLOT	PUSH	P
355D 79	0930		MOV	A,C ;CHECK COLUMN ON SCREEN
355E B7	0940		ORA	A
355F FA B4 35	0950		JM	OFFSCR
3562 78	0960		MOV	A,B ;CHECK ROW ON SCREEN
3563 B7	0970		ORA	A
3564 FA B4 35	0980		JM	OFFSCR
3567 FE 30	0990		CPI	48
3569 F2 B4 35	1000		JP	OFFSCR
356C C5	1010		PUSH	B ;ONOFF AND ROWCOL -> STA
356D AF	1020		XRA	A
356E 57	1030		MOV	D,A ;ROW -> DE
356F 58	1040		MOV	E,B
3570 01 03 00	1050		LXI	B,3
3573 CD 31 21	1060		CALL	DREM ;ROW % 3 -> DE, ROW / 3
3576 29	1070		DAD	H
3577 29	1080		DAD	H
3578 29	1090		DAD	H
3579 29	1100		DAD	H
357A 29	1110		DAD	H
357B 29	1120		DAD	H ;64 * (ROW/3) -> HL
357C C1	1130		POP	B
357D 79	1140		MOV	A,C
357E C6 00	1150		ADI	0 ;CLEARS CARRY
3580 1F	1160		RAR	;COL/2 -> A
3581 B5	1170		ORA	L
3582 6F	1180		MOV	L,A ;WORD = 64*(ROW/3)+COL/2
3583 79	1190		MOV	A,C ;COL -> A
3584 E6 01	1200		ANI	1 ;COL%2 -> A
3586 D6 01	1210		SUI	1 ;EITHER ALL 1'S OR ALL 0
3588 2F	1220		CMA	
3589 E6 03	1230		ANI	3 ;3*(COL%2) -> A
358B 83	1240		ADD	E ;ROW%3 + 3*(COL%2) -> A.
358C 5F	1250		MOV	E,A ; IS 5 - BIT POSITION;
358D AF	1260		XRA	A
358E 3E 20	1270		MVI	A,32 ;ZERO CARRY, SET BIT 5.

3590 1D	1280	LOOP	DCR	E	;RIGHT SHIFT THE BIT (E)
3591 FA 98 35	1290		JM	OUT	
3594 1F	1300		RAR		
3595 C3 90 35	1310		JMP	LOOP	
3598 5F	1320	OUT	MOV	E,A	;WORD WITH BIT -> E
3599 01 00 F8	1330		LXI	B,0F800H	
359C 09	1340		DAD	B	;MAKE WORD INTO VTI ADDR
359D 7E	1350		MOV	A,M	;MAKE SURE VTI BYTE IS A
359E B7	1360		ORA	A	; IS A GRAPHICS BYTE.
359F F2 A4 35	1370		JP	\$+2	
35A2 3E 3F	1380		MVI	A,63	
35A4 47	1390		MOV	B,A	;VTI BYTE -> B
35A5 F1	1400		POP	P	;ONOFF
35A6 B7	1410		ORA	A	
35A7 7B	1420		MOV	A,E	;WORD WITH BIT -> A
35A8 CA B0 35	1430		JZ	OFF	
35AB 2F	1440		CMA		;TURN SPOT ON
35AC A0	1450		ANA	B	
35AD 77	1460		MOV	M,A	
35AE AF	1470		XRA	A	;RETURN OK
35AF C9	1480		RET		
35B0 B0	1490	OFF	ORA	B	;TURN SPOT OFF
35B1 77	1500		MOV	M,A	
35B2 AF	1510		XRA	A	;RETURN OK
35B3 C9	1520		RET		
35B4 F1	1530	OFFSCR	POP	P	;OFF-SCREEN. RETURN 1 IN
35B5 3E 01	1540		MVI	A,1	
35B7 B7	1550		ORA	A	
35B8 C9	1560		RET		
35B9	1570				
35B9	1580				
35B9 CD B5 31	1590	MC1002	CALL	HOOKUP	
35BC C3 C5 21	1600		JMP	PZERO	
35BF CD CE 31	1610	MC1003	CALL	UNHOOK	
35C2 C3 C5 21	1620		JMP	PZERO	
35C5	1630				
35C5	1640				;PLOTS AN ENTIRE PICTURE ELEMENT, AS DEFINED IN
35C5	1650				; ARRAY OF 2N+1 BYTES. EACH PAIR OF BYTES IS RE
35C5	1660				; ROW, RELATIVE COLUMN IN RANGE +-127. LAST BYT
35C5	1670				; 128. RETURNS NUMBER OF POINTS PLOTTED ON SCR
35C5 57	1680	PLOTEL	MOV	D,A	;ON-OFF
35C6 AF	1690		XRA	A	
35C7 5F	1700		MOV	E,A	
35C8 3E 80	1710	PE2	MVI	A,128	
35CA BE	1720		CMP	M	
35CB 7B	1730		MOV	A,E	;RETURN ONSCREEN COUNT
35CC C8	1740		RZ		
35CD C5	1750		PUSH	B	;SAVE ABSOLUTES
35CE 78	1760		MOV	A,B	;ADD RELATIVES TO ABSOLU
35CF 86	1770		ADD	M	
35D0 47	1780		MOV	B,A	
35D1 23	1790		INX	H	
35D2 79	1800		MOV	A,C	
35D3 86	1810		ADD	M	
35D4 4F	1820		MOV	C,A	
35D5 23	1830		INX	H	

35D6 D5	1840	PUSH	D	;SAVE OTHER REG'S
35D7 E5	1850	PUSH	H	
35D8 7A	1860	MOV	A,D	;ON-OFF
35D9 CD 5C 35	1870	CALL	PLOT	
35DC E1	1880	POP	H	
35DD D1	1890	POP	D	
35DE C1	1900	POP	B	
35DF C2 E3 35	1910	JNZ	\$+1	;OFF-SCREEN
35E2 1C	1920	INR	E	;ON-SCREEN
35E3 C3 C8 35	1930	JMP	PE2	
35E6	1940			
35E6	1950			
35E6 CD 2E 20	1960	;LINKS TO PLOT		
35E9 D5	1970	MC1004 CALL	TOPTOI	
35EA CD 2E 20	1980	PUSH	D	
35ED D5	1990	CALL	TOPTOI	
35EE CD 2E 20	2000	PUSH	D	
35F1 D5	2010	CALL	TOPTOI	
35F2 CD 2E 20	2020	PUSH	D	
35F5 43	2030	CALL	TOPTOI	
35F6 D1	2040	MOV	B,E	;ABS ROW
35F7 4B	2050	POP	D	
35F8 D1	2060	MOV	C,E	;ABS ROW
35F9 7B	2070	POP	D	
35FA D1	2080	MOV	A,E	;ON/OFF FLAG
35FB EB	2090	POP	D	
35FC CD C5 35	2100	XCHG		;ELEMENT ARRAY -> HL
35FF 5F	2110	CALL	PLOT	
3600 AF	2120	MOV	E,A	;ON-SCREEN COUNT
3601 57	2130	XRA	A	
3602 C3 31 20	2140	MOV	D,A	
>TF		JMP	PUSHK	

VII. INSTALLING TINY-C ON YOUR PDP-11 SYSTEM

As furnished, the PDP-11 version of tiny-c, tiny-c/11, is ready to use with any DEC® monitor which handles the following programmed requests:

TTYIN
TTYOUT
LOOKUP
ENTER
READ
WRITE
CLOSE

The RT-11, RSX and RSTS monitors all satisfy this requirement. Since all device communication takes place through the tiny-c installation vector, users of CAPS-11, Paper Tape Software and IOX can assemble and install tiny-c after placing IOTs in the seven input/output entries in this vector and writing code to translate between the tiny-c peripheral specifications and IOX.

7.1 Logical Division of Tiny-c/11

Tiny-c/11 is divided logically into four parts: the interpreter, the interfaces, the MCs, and the Program Preparation System (PPS). The first three parts are collections of PDP-11 assembly language programs and subroutines. The last part, the PPS, is a tiny-c program together with a collection of tiny-c functions. The seven programmed requests mentioned above are found in the interface part of tiny-c/11.

® DEC is a trademark of Digital Equipment Corporation.

The interpreter consists of:

1. a main program which initiates the tiny-c interpreter and includes
 - a. assembly constants which size tiny-c
 - b. global declarations
 - c. tiny-c literals;
2. the tiny-c interpreter subroutines; and,
3. a data block consisting of tiny-c global variables and the installation vector.

The interfaces consist of all of tiny-c's input/output access routines and, hence, all monitor calls and programmed requests. The access routines are linked to the tiny-c interpreter through the installation vector. Users who have written their own input/output routines -- whether simple specification translation routines or full-fledged device drivers -- should place the addresses of these routines in the appropriate JMP instructions in the installation vector.

7.2 The Installation Vector

The tiny-c/11 installation vector is divided into three parts: seven input/output calls; four user option calls; and eight tiny-c/11 working area addresses.

7.2.1 Input/Output Routine Calls

The first seven entries in the tiny-c/11 installation vector are JMPs to the seven input/output routines used by tiny-c/11:

Installation Vector Entry =====	Subroutine Entry Point =====	Action =====
INCH	GETCHR	return a character from the console terminal right-justified (even-byte) in R0.
OUTCH	PUTCHR	transfer the character in the even-byte of R0 to the console terminal.
FLOAD	LOAD	load the file TLOAD.TCF into the tiny-c program space. Squeeze out nulls (ASCII 000) and carriage returns (ASCII 015).
FOPEN	OPEN	open the file described by the arguments placed on the machine stack as follows: 2(SP) 1 read 2 write 4(SP) address of 14-byte, fully-qualified file name 6(SP) file size 10(SP) channel or unit number return the status of the open in R0 as follows: 0 or greater file opened, return size of file in bytes -1 unit already open -2 file not found

FREAD

GETBL

read a block of data from the file system according to the arguments placed on the machine stack as follows:

2(SP) low order
address of
a 512-byte
data buffer

4(SP) channel or
unit data is
to be read
from

return the status of the read in R0 as follows:

0 or greater block read

-1 end-of-file

-2 read error

FWRITE

PUTBL

put a block of data to the file system according to the arguments placed on the machine stack as follows:

2(SP) address of
first byte
of the block

4(SP) address of
last address
of the block

6(SP) channel or
unit to re-
ceive the data

return the status of the write in R0 as follows:

0 or greater block written

-1 write error

FCLOSE CLOSE close the file described by the arguments placed on the machine stack as follows:

2(SP) channel or
 unit to be
 closed

FLOAD is intended to "bootstrap" the tiny-c system. It reads a specific file, TLOAD.TCF, from the system device. Since this file would typically have been prepared using the system editor, FLOAD provides a translation service between the output of this editor and the internal form required by the tiny-c interpreter. For most DEC systems this means simply eliminating carriage returns and nulls. FREAD and FWRITE, on the other hand, are meant to be complementary. That is, FREAD returns exactly what FWRITE has written.

7.2.2 User Option Routines

There are four opportunities for user routine activity during the execution of the tiny-c interpreter. JMP instructions to user routines to be called at each of these opportunities are the next four entries in the tiny-c/11 installation vector. The installation vector and corresponding opportunity description are as follows:

USERMC If an MC call with function number greater than 1000 is encountered in the tiny-c program being executed, its arguments are placed on the machine stack and a JSR to USERMC is executed as described in Section 2.10.

PRBEGN Before the standard Machine Call eleven, MC 11, enters an application program, a JSR to PRBEGN is executed.

STBEGN Before the interpretation of each tiny-c statement, a JSR to STBEGN is executed.

PRDONE After a return from an application program entered through the standard Machine Call eleven, MC 11, but before the tiny-c interpreter continues to process the program, a JSR to PRDONE is executed. which entered the application.

7.2.3 Working Areas

The four tiny-c working areas described in Chapter V have the addresses of their first and last bytes recorded in the next eight entries in the installation vector, as follows:

BSTACK	beginning of tiny-c stack
ESTACK	end of tiny-c stack
BVAR	beginning of variable table
EVAR	end of variable table
BFUN	beginning of function table
EFUN	end of function table
BPR	beginning of program space
EPR	end of program space

The ninth and last entry in this final section of the installation vector is

MSTACK The beginning (high-order) address of the machine stack space to be used by tiny-c. This address is loaded into SP when tiny-c is started.

7.3 The Sizing Constants

There are six assembly constants which can be used to control the size of the tiny-c/11 working areas. Their names, default values, and descriptions are as follows:

Name ====	Default Value =====	Description =====
MAXSTACK	30	the maximum number of entries on the tiny-c stack.
MAXVAR	100	the maximum number of active variable and function names in a tiny-c program; includes both PPS and application program variables.
MAXFUN	30	the maximum number of functions which can be active (not defined) in a tiny-c program. Each recursion instance counts as another active function.
MAXPR	20000	the size in bytes of the tiny-c program space.
MAXDEPTH	50	the maximum subroutine call depth of the tiny-c interpreter. Twelve times this value is the size of the machine stack allocated to tiny-c.
VLEN	8	the size in bytes of canonicalized variable and function names.

7.4 Subroutine Call and Processor Stack Regimen

Tiny-c/11 uses the same subroutine call and processor stack regimen as that produced by the C compiler. This regimen is implemented by two subroutines, SAVE and RETURN. SAVE is called upon entry to any subroutine as follows:

```
JSR      R5,SAVE
```

and RETURN is called to leave a subroutine as follows:

```
JMP      RETURN
```


This method of subroutine entry and exit has the following advantages:

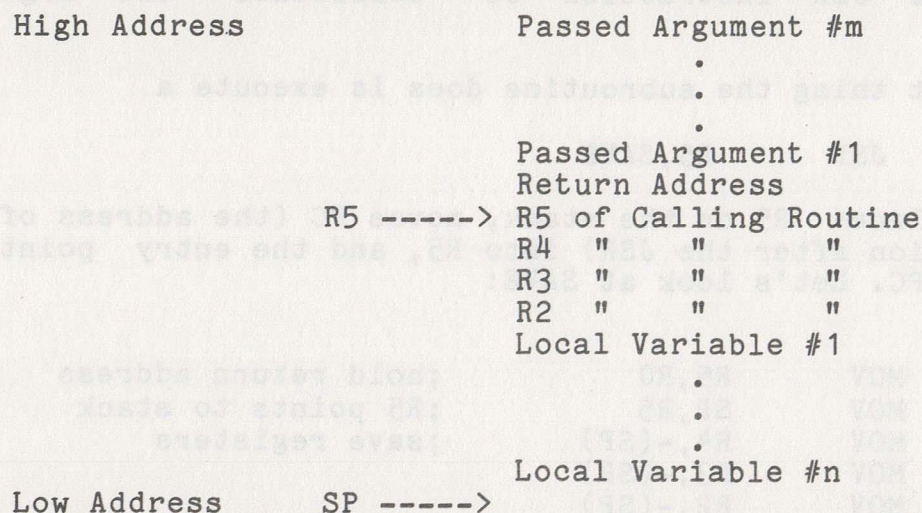
1. safety of registers R2, R3, R4, and SP over all subroutines which follow the regimen;
2. allocation, preservation, and deallocation of local variables on the processor stack;
3. recursive subroutine calling;
4. consistent handling of subroutine arguments; and,
5. a stack which contains much helpful debugging information.

It also has the following disadvantages:

1. reservation of register R5 as an additional stack pointer;
2. an unnecessarily large stack area; and
3. potentially wasted save, access, and restore cycles.

7.4.1 Local Stack Structure

Locally -- that is, within a subroutine -- the structure of the stack maintained by SAVE and RETURN looks like this:



Thus, arguments passed into the called subroutine are addressed as 4(R5), 6(R5), ... and local variables are addressed as -10(R5), -12(R5), ... Upon return from SAVE, the stack pointer points to the address under the saved R2 value. The called subroutine makes room for its own local variables by simply subtracting their number from SP.

7.4.2 Subroutine Calling

A subroutine is called by simply placing its arguments on the stack and executing a JSR to it. Thus, for example, to push a 0 on the tiny-c stack, the following code is executed:

```

CLR      (SP)
MOV      #2,-(SP)
MOV      #101,-(SP)
CLR      -(SP)
JSR      PC,@#PUSH

```


In order to conserve on stack space, SP should be reset upon return from the called subroutine. To continue our example,

```
ADD    #6,SP
```

after the JSR instruction to "deallocate" the argument space.

The first thing the subroutine does is execute a

```
JSR    R5,SAVE
```

which places R5 on the stack, moves PC (the address of the instruction after the JSR) into R5, and the entry point to SAVE in PC. Let's look at SAVE:

```
SAVE:  MOV    R5,R0          ;hold return address
        MOV    SP,R5         ;R5 points to stack
        MOV    R4,-(SP)      ;save registers
        MOV    R3,-(SP)
        MOV    R2,-(SP)
        JSR    PC,(R0)       ;go back to (R0)
```

The JSR to SAVE pushed the calling routine's R5 reference pointer and the new R5 points to it in the stack.

After the JSR to SAVE, the subroutine can freely use R2, R3, R5, and SP. To return control to the routine that called this subroutine, execute a

```
JMP    RETURN
```

to the following code:

```
RETURN: MOV    R5,R2          ;temporary copy of R5
        MOV    -(R2),R4       ;restore registers
        MOV    -(R2),R3
        MOV    -(R2),R2
        MOV    R5,SP          ;point to old R5
        MOV    (SP)+,R5       ;restore R5 and point to
                                ; return
        RTS    PC             ;go to return and bump SP
```


The calling routine gets registers R2, R3, R4, R5, and SP back in exactly the same shape it left them. The called routine can pass values back to the calling routine either in R0 and R1 or in the arguments. Typically, a called subroutine will not alter the values of a calling subroutine's local variables unless it is given a pointer argument. Thus, the relationship between local variables and arguments which is enforced tiny-c language is echoed in the in the tiny-c interpreter subroutines.

7.4.3 Interpreter Subroutines Following the Regimen

One could, of course, require that all subroutines observe the SAVE/RETURN regimen. Subroutines, such as ST, which can indirectly call themselves, clearly MUST follow the regimen. Others, such as BLANKS, which don't call any other routines, simply waste cycles doing needless saves and restores. The following tabulation separates those subroutines which need the regimen from those which don't:

FIGURE 7-1

DO		SUBROUTINES WHICH		DO NOT	
		USE SAVE and RETURN			
=====					
ADDRVAL	LINK	ALPHA	NEWFUN		
ASGN	NEWVAR	ALPHAN	NUM		
CANON	RELN	ATOI	POP		
CONST	SETARG	BLANKS	PUSH		
ENTER	SKIPST	CEQ	REM		
EQ	ST	CEQN	SKIP		
EXPR	TERM	ESET	SYMNAM		
FACTOR	VALLOC	FUNDON	TCTOI		
		LIT	TOPTOI		
		MOVN			

End of FIGURE 7-1

Further improvements are certainly possible. While tiny-c/11 has made some concessions to speed and efficiency, program consistency and clarity have not been sacrificed to obtain them. Owners will find that it is easier to optimize an unoptimized program than it is to reoptimize a misoptimized one.

Subroutine's local variables unless it is given a pointer argument. Thus, the relationship between local variables and arguments which is enforced in tiny-c language is relaxed in the tiny-c interpreter subroutines.

7.4.3 Interpreter Subroutines Following the Register

One could, of course, require that all subroutines observe the SAVE/RETURN register. Subroutines, such as ST, which can indirectly call themselves, clearly MUST follow the register. Others, such as BLANKS, which don't call any other routines, simply waste cycles doing needless saves and restores. The following tabulation separates those subroutines which need the register from those which don't:

FIGURE 7-1

SUBROUTINES WHICH			
DO NOT			
USE SAVE and RETURN			

NEWVAR	ALPHA	LINK	ADDRESS
NUM	ALPHAN	NEWVAR	ASGN
FOR	ATOL	RELW	CANON
TUSH	BLANKS	SETARG	CONST
REM	CEQ	EXIPST	ENTER
SELF	CCW	ST	ED
SYMMAN	EBET	TERN	EXPR
TOTOT	EDNDON	VALJOC	FACTOR
TOPTOT	LIT		
	MOVH		

BIBLIOGRAPHY

- Feurzeig, W., et al., "Programming Languages as a Conceptual Framework for Teaching Mathematics," BBN Report No. 2165 (June 1971).
- Gries, David, "On Structured Programming - A Reply to Smoliar," CACM, Vol. 17, No. 11 (November 1974).
- Kemeny, J. G. and Kurtz, T. E., BASIC PROGRAMMING, Wiley, 1967.
- Kernighan, Brian W. and Plauger, P. J., SOFTWARE TOOLS, Addison-Wesley, 1967.
- Kernighan, Brian W. and Ritchie, Dennis M., THE C PROGRAMMING LANGUAGE, Prentice-Hall Software Series, 1978.
- Madden, J. Gregory, "C: A Language for Microprocessors?," BYTE, Vol. 2, No. 10 (October 1977).
- Ritchie, D. M. and Thompson, K., "The UNIX Time-Sharing System," CACM, Vol. 17, No. 3 (March 1974).
- Ritchie, D. M., et al., "The C Programming Language," Computer Science Technical Report No. 31, Bell Telephone Laboratories, Murray Hill, N. J. 07974 (October 1975).
- Snyder, Alan, "A Portable Compiler for the Language C," MIT Project MAC Technical Report 149, May 1975.

BIBLIOGRAPHY

1. Faurst, W., et al., "Programming Languages as a Conceptual Framework for Teaching Mathematics," IBM Report No. 2102 (June 1971).
2. Gries, David, "On Structured Programming - A Reply to Smollar," CACM, Vol. 17, No. 11 (November 1974).
3. Kennedy, J. G. and Kurtz, J. E., BASIC PROGRAMMING, Wiley, 1967.
4. Kernighan, Brian W. and Plauger, P. J., SOFTWARE TOOLS, Addison-Wesley, 1987.
5. Kernighan, Brian W. and Ritchie, Dennis M., THE C PROGRAMMING LANGUAGE, Prentice-Hall Software Series, 1978.
6. Madson, J. Gregory, "C: A Language for Microprocessors," BYTE, Vol. 2, No. 10 (October 1977).
7. Ritchie, D. M. and Thompson, K., "The UNIX Time-Sharing System," CACM, Vol. 17, No. 7 (March 1974).
8. Ritchie, D. M., et al., "The C Programming Language," Computer Science Technical Report No. 31, Bell Telephone Laboratories, Murray Hill, N. J. 07814 (October 1972).
9. Snyder, Alan, "A Portable Compiler for the Language C," MIT Project MAC Technical Report 149, May 1975.

A

tiny-c/8080 Version 80-01-01

2000			0000		;error codes	
2000			0010		STATERR EQU	1
2000			0020		CURSERR EQU	2
2000			0030		SYMERR EQU	3
2000			0040		RPARERR EQU	5
2000			0050		RANGERR EQU	6
2000			0060		CLASERR EQU	7
2000			0070		SYNXERR EQU	9
2000			0080		LVALERR EQU	14
2000			0090		PUSHERR EQU	16
2000			0100		TMFUERR EQU	17
2000			0110		TMVRERR EQU	18
2000			0120		TMVLERR EQU	19
2000			0130		LINKERR EQU	20
2000			0140		ARGSERR EQU	21
2000			0150		LBRCERR EQU	22
2000			0160		MCERR EQU	24
2000			0170		SYMERRA EQU	26
2000			0180		KILL EQU	99
2000			0190		;recognition length of symbols	
2000			0200		VLEN EQU	8
2000			0210		;end-of-line character	
2000			0220		ASCRET EQU	0DH
2000			0230		;	
2000			0240		;entry points	
2000	C3	58	26		JMP	COLD
2003	C3	7E	26		JMP	WARM
2006	C3	81	26		JMP	HOT
2009						
2009	01				;tailoring vector	
200A	C3	00	00	U	ECHO DB	1 ;zero suppresses char echo
200D	C3	00	00	U	INCH JMP	X
2010	C3	00	00	U	OUTCH JMP	X
2013	C3	00	00	U	CHRDY JMP	X
2016	C3	00	00	U	FOPEN JMP	X
2019	C3	00	00	U	FREAD JMP	X
201C	C3	00	00	U	FWRITE JMP	X
201F	C3	00	00	U	FCLOSE JMP	X
					USERMC JMP	MCESET

2022 00	0380	PRBEGIN	NOP	
2023 00	0390		NOP	
2024 C9	0400		RET	
2025 00	0410	STBEGIN	NOP	
2026 00	0420		NOP	
2027 C9	0430		RET	
2028 00	0440	PRDONE	NOP	
2029 00	0450		NOP	
202A C9	0460		RET	
202B	0470	;MC tools		
202B C3 12 2F	0480	XMCESET	JMP	MCESET
202E C3 83 21	0490	XTOPTOI	JMP	TOPTOI
2031 C3 C8 21	0500	XPUSHK	JMP	PUSHK
2034 00	0510	MCARGS	DB	0
2035	0520	;escape character		
2035 1B	0530	ESCAPE	DB	1BH
2036	0540	;space allocation		
2036 00 38	0550	BSTACK	DW	3800H
2038 10 C7	0560	ESTACK	DW	-38F0H
203A 00 39	0570	BFUN	DW	3900H
203C 10 C6	0580	EFUN	DW	-39F0H
203E 00 3A	0590	BVAR	DW	3A00H
2040 10 C2	0600	EVAR	DW	-3DF0H
2042 00 3E	0610	BPR	DW	3E00H
2044 10 A0	0620	EPR	DW	-5FF0H
2046 00 00	0630	MSTACK	DW	0
2048	0640	;standard cells		
2048 00 00	0650	ERR	DW	0
204A 00 00	0660	ERRAT	DW	0
204C 00	0670	LEAVE	DB	0
204D 00	0680	BRAKE	DB	0
204E 00 00	0690	TOP	DW	0
2050 00 00	0700	NXTVAR	DW	0
2052 00 00	0710	CURFUN	DW	0
2054 00 00	0720	CURGLBL	DW	0
2056 00 00	0730	FNAME	DW	0
2058 00 00	0740	LNAME	DW	0
205A 00 00	0750	STCURS	DW	0

CALLED AT 2CCFH

CALLED AT 2908H. good for TRACE, etc

CALLED AT 2C05H

tiny-c/8080 Version 80-01-01

205C 00 00	0760	CURSOR	DW	0	
205E 00 00	0770	PRUSED	DW	0	
2060 00 00	0780	PROGEND	DW	0	;stored negative
2062 00	0790	APPLVL	DB	0	
2063	0800				
2063	0810				
2063	0820	BALPHS	EQU	\$	;beginning of alphabetics
2063 69	0830	XIF	DB	'i'	
2064 66	0840		DB	'f'	
2065 00	0850		DB	0	
2066 65	0860	XELS	DB	'e'	
2067 6C	0870		DB	'l'	
2068 73	0880		DB	's'	
2069 65	0890		DB	'e'	
206A 00	0900		DB	0	
206B 69	0910	XINT	DB	'i'	
206C 6E	0920		DB	'n'	
206D 74	0930		DB	't'	
206E 00	0940		DB	0	
206F 63	0950	XCHAR	DB	'c'	
2070 68	0960		DB	'h'	
2071 61	0970		DB	'a'	
2072 72	0980		DB	'r'	
2073 00	0990		DB	0	
2074 77	1000	XWHI	DB	'w'	
2075 68	1010		DB	'h'	
2076 69	1020		DB	'i'	
2077 6C	1030		DB	'l'	
2078 65	1040		DB	'e'	
2079 00	1050		DB	0	
207A 72	1060	XRET	DB	'r'	
207B 65	1070		DB	'e'	
207C 74	1080		DB	't'	
207D 75	1090		DB	'u'	
207E 72	1100		DB	'r'	
207F 6E	1110		DB	'n'	
2080 00	1120		DB	0	
2081 62	1130	XBRK	DB	'b'	

tiny-c/8080 Version 80-01-01

2082	72	1140	DB	'r'	
2083	65	1150	DB	'e'	
2084	61	1160	DB	'a'	
2085	6B	1170	DB	'k'	
2086	00	1180	DB	0	
2087	65	1190	XENDL DB	'e'	
2088	6E	1200	DB	'n'	
2089	64	1210	DB	'd'	
208A	6C	1220	DB	'l'	
208B	69	1230	DB	'i'	
208C	62	1240	DB	'b'	
208D	72	1250	DB	'r'	
208E	61	1260	DB	'a'	
208F	72	1270	DB	'r'	
2090	79	1280	DB	'y'	
2091	00	1290	DB	0	
2092	72	1300	XR DB	'r'	;loader 'read' command
2093	67	1310	XG DB	'g'	;'go' command
2094	FF	1320	DB	-1	;end of alphabetics
2095	5B	1330	LB DB	'['	
2096	00	1340	DB	0	
2097	5D	1350	RB DB	']'	
2098	00	1360	DB	0	
2099	28	1370	LPAR DB	'('	
209A	00	1380	DB	0	
209B	29	1390	RPAR DB	')'	
209C	00	1400	DB	0	
209D	2C	1410	COMMA DB	','	
209E	00	1420	DB	0	
209F	0D	1430	NEWLINE DB	ASCRT	
20A0	00	1440	DB	0	
20A1	2F	1450	CMNT DB	'/'	
20A2	2A	1460	XSTAR DB	'*'	
20A3	00	1470	DB	0	
20A4	3B	1480	SEMI DB	','	
20A5	00	1490	DB	0	
20A6	25	1500	XPCNT DB	'%'	
20A7	00	1510	DB	0	

tiny-c/8080 Version 80-01-01

20A8 2F	1520	XSLASH	DB	'/'
20A9 00	1530		DB	0
20AA 2B	1540	XPLUS	DB	'+'
20AB 00	1550		DB	0
20AC 2D	1560	XMINUS	DB	'-'
20AD 00	1570		DB	0
20AE 3C	1580	LT	DB	'<'
20AF 00	1590		DB	0
20B0 3E	1600	GT	DB	'>'
20B1 00	1610		DB	0
20B2 21	1620	NOTEQ	DB	'!'
20B3 3D	1630		DB	'='
20B4 00	1640		DB	0
20B5 3D	1650	EQEQ	DB	'='
20B6 3D	1660	XEQ	DB	'='
20B7 00	1670		DB	0
20B8 3E	1680	GE	DB	'>'
20B9 3D	1690		DB	'='
20BA 00	1700		DB	0
20BB 3C	1710	LE	DB	'<'
20BC 3D	1720		DB	'='
20BD 00	1730		DB	0
20BE 0D	1740	XNL	DB	ASCRT
20BF 00	1750		DB	0
20C0	8888	END1	EQU	\$

20C0	0000	;EQ performs an assignment of top into top-1. Top-1
20C0	0010	; must be an lvalue.
20C0 CD 83 21	0020	EQ CALL TOPTOI ;value into DE
20C3 D5	0030	PUSH D ;stuff to be assigned
20C4 CD A9 21	0040	CALL POP ;where to assign
20C7 B7	0050	ORA A
20C8 CA CD 20	0060	JZ EQ2 ;if class>0 set size=2
20CB 0E 02	0070	MVI C,2
20CD 78	0080	EQ2 MOV A,B ;must be lvalue
20CE FE 4C	0090	CPI 'L'
20D0 C2 DF 20	0100	JNZ EQERR
20D3 EB	0110	XCHG ;where -> HL
20D4 D1	0120	POP D ;stuff -> DE
20D5 73	0130	MOV M,E ;assign lo byte
20D6 0D	0140	DCR C ;size--
20D7 CA C8 21	0150	JZ PUSHK ;call/ret, put result on stack
20DA 23	0160	INX H
20DB 72	0170	MOV M,D ;hi byte
20DC C3 C8 21	0180	JMP PUSHK ;call/ret, put result on stack
20DF CD F1 21	0190	EQERR CALL ESET
20E2 0E	0200	DB LVALERR
20E3 D1	0210	POP D
20E4 C3 C8 21	0220	JMP PUSHK ;skip the assign part
20E7	0230	;
20E7	0240	;- (BC) -> BC
20E7 79	0250	DNEG MOV A,C
20E8 2F	0260	CMA
20E9 4F	0270	MOV C,A
20EA 78	0280	MOV A,B
20EB 2F	0290	CMA
20EC 47	0300	MOV B,A
20ED 03	0310	INX B
20EE C9	0320	RET
20EF	0330	;
20EF	0340	;difference between two top values -> DE, setting Z, CY
20EF CD A0 21	0350	TOPDIF CALL POPTWO ;hence fall into DSUB.
20F2	0360	;
20F2	0370	; (DE) - (BC) -> DE

00 00

ØF 21

1F 21

17 21

• • • • •

28 21

09 21

211F		0760		; shift BC right, setting Z if 0.
211F AF		0770	BCRS XRA A	;zero CY flag
2120 78		0780	MOV A,B	
2121 1F		0790	RAR	
2122 47		0800	MOV B,A	
2123 79		0810	MOV A,C	
2124 1F		0820	RAR	;picks up carry left by hi byte
2125 4F		0830	MOV C,A	
2126 B0		0840	ORA B	
2127 C9		0850	RET	
2128		0860		; shift DE left. Sets z iff (DE)==0.
2128 AF		0870	DELS XRA A	;zero CY flag
2129		0880		; rotate DE left, CY -> lo bit
2129 7B		0890	RDEL MOV A,E	;lo byte first
212A 17		0900	RAL	
212B 5F		0910	MOV E,A	
212C 7A		0920	MOV A,D	
212D 17		0930	RAL	;picks up carry left by lo byte
212E 57		0940	MOV D,A	
212F B3		0950	ORA E	
2130 C9		0960	RET	
2131		0970		
2131		0980		; (DE) % (BC) -> DE, quotient in HL.
2131 7A		0990	DREM MOV A,D	;sign of result -> stack
2132 A8		1000	XRA B	
2133 F5		1010	PUSH P	
2134 7A		1020	MOV A,D	;make factors positive
2135 B7		1030	ORA A	
2136 FC 74 21		1040	CM DENEG	
2139 78		1050	MOV A,B	
213A B7		1060	ORA A	
213B FC E7 20		1070	CM DNEG	
213E 3E 10		1080	MVI A,16	;shift count -> stack
2140 F5		1090	PUSH P	
2141 EB		1100	XCHG	;numerator -> HL
2142 11 00 00		1110	LXI D,0	;partial remainder -> DE
2145 CD 81 21		1120	D2 CALL HLLS	;divide loop. Long left shift
2148 CD 29 21		1130	CALL RDEL	; DEHL.

tiny-c/8080 Version 80-01-01

214B CA 5B 21	1140	JZ	D3	
214E CD 7C 21	1150	CALL	DCMP	;test BC <= DE
2151 FA 5B 21	1160	JM	D3	
2154 7D	1170	MOV	A,L	;set lo bit of L, and subtract
2155 F6 01	1180	ORI	1	; divisor from partial
2157 6F	1190	MOV	L,A	; remainder
2158 CD F2 20	1200	CALL	DSUB	
215B F1	1210	POP	P	;decrement shift count
215C 3D	1220	DCR	A	
215D CA 64 21	1230	JZ	D4	
2160 F5	1240	PUSH	P	
2161 C3 45 21	1250	JMP	D2	
2164 F1	1260	POP	P	;put sign on quotient and rem
2165 F0	1270	RP		
2166 CD 74 21	1280	CALL	DENEG	
2169 EB	1290	XCHG		
216A CD 74 21	1300	CALL	DENEG	
216D EB	1310	XCHG		
216E C9	1320	RET		
216F	1330			
216F	1340			
216F CD 31 21	1350	DDIV	CALL DREM	
2172 EB	1360	XCHG		
2173 C9	1370	RET		
2174	1380			
2174	1390			
2174 7A	1400	DENEG	MOV A,D	
2175 2F	1410	CMA		
2176 57	1420	MOV	D,A	
2177 7B	1430	MOV	A,E	
2178 2F	1440	CMA		
2179 5F	1450	MOV	E,A	
217A 13	1460	INX	D	
217B C9	1470	RET		
217C	1480			
217C	1490			
217C	1500			
217C 7B	1510	DCMP	MOV A,E	

217D 91	1520	SUB	C
217E 7A	1530	MOV	A,D
217F 98	1540	SBB	B
2180 C9	1550	RET	
2181	1560	;	
2181	1570	;HL left shift	
2181 29	1580	HLLS DAD	H
2182 C9	1590	RET	
2183	1600	;	
2183	1610	;@@@@@@@@ stack tools @@@@@@@@@@	
2183	1620	;	
2183	1630	;TOPTOI pops top of stack into DE, converting lvalue	
2183	1640	; to actual if necessary.	
2183 CD A9 21	1650	TOPTOI CALL POP	;class in A, lvalue in B,
2186 32 9F 21	1660	STA	TPCLASS ; size in C, 'stuff' in DE
2189 78	1670	MOV	A,B
218A FE 41	1680	CPI	'A'
218C CA 93 21	1690	JZ	TT2
218F EB	1700	XCHG	;fetch data
2190 5E	1710	MOV	E,M
2191 23	1720	INX	H
2192 56	1730	MOV	D,M
2193 0D	1740	TT2 DCR	C ;if size 1 and class 0 return
2194 C0	1750	RNZ	; lo byte, with sign propagated
2195 3A 9F 21	1760	LDA	TPCLASS ; thru hi byte.
2198 B7	1770	ORA	A
2199 C0	1780	RNZ	
219A 7B	1790	MOV	A,E
219B 07	1800	RLC	;propagate sign into D.
219C 9F	1810	SBB	A
219D 57	1820	MOV	D,A
219E C9	1830	RET	
219F 00	1840	TPCLASS DB	0
21A0	1850	;	
21A0	1860	;pops two from stack, top -> bc, next -> de.	
21A0 CD 83 21	1870	POPTWO CALL	TOPTOI
21A3 D5	1880	PUSH	D
21A4 CD 83 21	1890	CALL	TOPTOI

tiny-c/8080 Version 80-01-01

21A7 C1	1900	POP	B	
21A8 C9	1910	RET		
21A9	1920			
21A9	1930	; pops the stack into A, B, C, DE. New top in HL.		
21A9 2A 4E 20	1940	POP	LHLD	TOP
21AC 7E	1950	MOV	A,M	;class
21AD 23	1960	INX	H	
21AE 46	1970	MOV	B,M	;lvalue
21AF 23	1980	INX	H	
21B0 4E	1990	MOV	C,M	;size
21B1 23	2000	INX	H	
21B2 5E	2010	MOV	E,M	;stuff, lo-byte
21B3 23	2020	INX	H	
21B4 56	2030	MOV	D,M	;stuff, hi-byte
21B5 C5	2040	PUSH	B	
21B6 01 F7 FF	2050	LXI	B,-9	
21B9 09	2060	DAD	B	;decrement top by 5.
21BA C1	2070	POP	B	
21BB 22 4E 20	2080	SHLD	TOP	
21BE C9	2090	RET		
21BF	2100			
21BF	2110	; pushes constant 1.		
21BF 11 01 00	2120	PONE	LXI	D,1
21C2 C3 C8 21	2130		JMP	PUSHK
21C5	2140	; pushes constant 0.		
21C5 11 00 00	2150	PZERO	LXI	D,0
21C8	2160	; pushes constant in DE		
21C8 AF	2170	PUSHK	XRA	A ;class 0
21C9 06 41	2180		MVI	B,'A' ;actual
21CB 0E 02	2190		MVI	C,2 ;2 byte size
21CD	2200	; pushes class (A), lvalue (B), size (C), stuff (DE)		
21CD	2210	; onto stack.		
21CD 2A 4E 20	2220	PUSH	LHLD	TOP ;add 5 to top.
21D0 D5	2230		PUSH	D
21D1 11 05 00	2240		LXI	D,5
21D4 19	2250		DAD	D
21D5 22 4E 20	2260		SHLD	TOP
21D8 EB	2270		XCHG	

tiny-c/8080 Version 80-01-01

21D9 2A 38 20	2280	LHLD	ESTACK	
21DC 19	2290	DAD	D	
21DD EB	2300	XCHG		;top -> HL
21DE DA EC 21	2310	JC	PERR	
21E1 D1	2320	POP	D	;restore stuff
21E2 77	2330	MOV	M,A	
21E3 23	2340	INX	H	
21E4 70	2350	MOV	M,B	
21E5 23	2360	INX	H	
21E6 71	2370	MOV	M,C	
21E7 23	2380	INX	H	
21E8 73	2390	MOV	M,E	
21E9 23	2400	INX	H	
21EA 72	2410	MOV	M,D	
21EB C9	2420	RET		
21EC CD F1 21	2430	PERR CALL	ESET	
21EF 10	2440	DB	PUSHERR	
21F0 C9	2450	RET		
21F1	2460			
21F1	2470			
21F1 3A 48 20	2480	ESET LDA	ERR	; @@@@ ESET sets ERR unless one is already set @@@@
21F4 E3	2490	XTHL		
21F5 B7	2500	ORA	A	
21F6 CA FC 21	2510	JZ	ES2	
21F9 23	2520	INX	H	
21FA E3	2530	XTHL		
21FB C9	2540	RET		
21FC 7E	2550	ES2 MOV	A,M	
21FD 23	2560	INX	H	
21FE E3	2570	XTHL		
21FF 32 48 20	2580	STA	ERR	
2202 2A 5C 20	2590	LHLD	CURSOR	
2205 22 4A 20	2600	SHLD	ERRAT	
2208 C9	2610	RET		
2209	2620			
2209	2630			;store 0's from (DE) thru (HL) inclusive
2209 06 00	2640	ZERO MVI	0,B	
220B	2650			;store (B) from (DE) thru (HL) inclusive

tiny-c/8080 Version 80-01-01

220B 7D	2660	BZAP	MOV	A,L
220C 93	2670		SUB	E
220D 7C	2680		MOV	A,H
220E 9A	2690		SBB	D
220F D8	2700		RC	
2210 70	2710		MOV	M,B
2211 2B	2720		DCX	H
2212 C3 0B 22	2730		JMP	BZAP
2215	2740			
2215	2750			
2215 7E	2760	PS	MOV	A,M
2216 B7	2770		ORA	A
2217 C8	2780		RZ	
2218 CD 0D 20	2790		CALL	OUTCH
221B 23	2800		INX	H
221C C3 15 22	2810		JMP	PS
221F	8888	END2	EQU	\$

;print string starting at (HL), terminated by null byte


```

221F      0000      ;
221F      0010      ;@@@@@@@@@ SCAN TOOLS @@@@@@@@@@@@@@
221F      0020      ;
221F      0030      ;LIT is used to match literals. It advances the cursor
221F      0040      ; over blanks, then attempts a match with the literal.
221F      0050      ; DE points to the literal, which is terminated by a
221F      0060      ; null byte. On match, the cursor is advanced
221F      0070      ; beyond the matched text, and NZ is set. On no match
221F      0080      ; the cursor is not advanced (except over the initial
221F      0090      ; blanks), and Z is set. LIT is called often, so some
221F      0100      ; attention to speed is given, mainly by using inline
221F      0110      ; code for blanks and string matching.
221F 2A 5C 20      0120 LIT      LHL      D      CURSOR
2222 3E 20          0130          MVI      A,' '      ;trim blanks
2224 BE            0140 LIT2     CMP      M
2225 C2 2C 22      0150          JNZ      LIT3
2228 23            0160          INX      H
2229 C3 24 22      0170          JMP      LIT2
222C 22 5C 20      0180 LIT3     SHLD     CURSOR      ;capture cursor, in case no mch
222F 1A            0190 LIT4     LDAX     D      ;char from literal
2230 B7            0200          ORA      A
2231 CA 3D 22      0210          JZ       MATCH      ;null signals end of literal
2234 BE            0220          CMP      M      ;char from program
2235 13            0230          INX      D
2236 23            0240          INX      H
2237 CA 2F 22      0250          JZ       LIT4
223A AF            0260          XRA      A      ;no match, return Zero
223B B7            0270          ORA      A
223C C9            0280          RET
223D 22 5C 20      0290 MATCH    SHLD     CURSOR      ;capture new cursor
2240 2F            0300          CMA
2241 B7            0310          ORA      A      ;return Not Zero
2242 C9            0320          RET
2243              0330      ;
2243              0340      ;advances cursor over blanks. Puts cursor in HL.
2243 2A 5C 20      0350 BLANKS   LHL      D      CURSOR
2246 3E 20          0360          MVI      A,' '
2248 BE            0370 LOOP     CMP      M

```


tiny-c/8080 Version 80-01-01

2249	C2	50	22	0380	JNZ	OUT	
224C	23			0390	INX	H	
224D	C3	48	22	0400	JMP	LOOP	
2250	22	5C	20	0410	OUT	SHLD	CURSOR
2253	C9			0420		RET	
2254				0430			
2254				0440			
2254				0450			
2254				0460			
2254				0470			
2254	16	01		0480	SKIP	MVI	D,1 ;counter
2256	7E			0490	SK2	MOV	A,M
2257	B8			0500		CMP	B
2258	CA	6A	22	0510		JZ	SKL ;match left delimiter
225B	B9			0520		CMP	C
225C	C2	6B	22	0530		JNZ	SKNEXT
225F	15			0540		DCR	D ;match right delimiter
2260	C2	6B	22	0550		JNZ	SKNEXT
2263	23			0560		INX	H ;all done, bump over last
2264	22	5C	20	0570		SHLD	CURSOR ; matched.
2267	37			0580		STC	
2268	3F			0590		CMC	
2269	C9			0600		RET	;CY off on success
226A	14			0610	SKL	INR	D
226B	23			0620	SKNEXT	INX	H ;bump HL, test for overflow
226C	EB			0630		XCHG	;cursor -> DE
226D	E5			0640		PUSH	H ;make H safe
226E	2A	60	20	0650		LHLD	PROGEND ;stored negative, so add
2271	19			0660		DAD	D
2272	E1			0670		POP	H
2273	EB			0680		XCHG	;now all reg's restored
2274	D2	56	22	0690		JNC	SK2
2277	CD	F1	21	0700		CALL	ESET
227A	02			0710		DB	CURSERR
227B	37			0720		STC	
227C	C9			0730		RET	;CY set on error
227D				0740			
227D				0750			

;tests if (A) is alphanumeric. Plus on yes.

227D	FE	30	0760	ALNUM	CPI	'0'	
227F	F8		0770		RM		
2280	FE	3A	0780		CPI	'9'+1	
2282	FA	98 22	0790		JM	YESA	
2285			0800	;tests if (A) is alpha. Plus on yes.			
2285	FE	41	0810	ALPHA	CPI	'A'	
2287	F8		0820		RM		;not alpha
2288	FE	5B	0830		CPI	'Z'+1	
228A	FA	98 22	0840		JM	YESA	
228D	FE	61	0850		CPI	'a'	
228F	F8		0860		RM		
2290	FE	7B	0870		CPI	'z'+1	
2292	FA	98 22	0880		JM	YESA	
2295	2F		0890		CMA		;not alpha, this sets Minus.
2296	B7		0900		ORA	A	
2297	C9		0910		RET		
2298	AF		0920	YESA	XRA	A	;set Plus.
2299	C9		0930		RET		
229A			0940				
229A			0950	;matches a variable or function name. Sets FNAME,			
229A			0960	; LNAME to first and last chars of the name. Returns			
229A			0970	; Not Zero on match, Zero on no match.			
229A	CD	43 22	0980	SYMNAME	CALL	BLANKS	
229D	22	56 20	0990		SHLD	FNAME	
22A0	7E		1000		MOV	A,M	
22A1	CD	85 22	1010		CALL	ALPHA	
22A4	FA	B7 22	1020		JM	SY3	
22A7	23		1030	SY2	INX	H	;is a symbol, find its end.
22A8	7E		1040		MOV	A,M	
22A9	CD	7D 22	1050		CALL	ALNUM	
22AC	F2	A7 22	1060		JP	SY2	
22AF	22	5C 20	1070		SHLD	CURSOR	;just beyond symbol
22B2	2B		1080		DCX	H	
22B3	22	58 20	1090		SHLD	LNAME	;symbol end
22B6	C9		1100		RET		
22B7	AF		1110	SY3	XRA	A	;no symbol, return Z
22B8	C9		1120		RET		
22B9			1130				

Address	Hex	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418
---------	-----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

2304	D2	F3	22	1520	JNC	CN7	
2307	C3	3E	23	1530	JMP	CNERR	;cursor overflow
230A	77			1540	MOV	M,A	;end quote found, replace with
230B	2B			1550	DCX	H	; a null.
230C	22	58	20	1560	SHLD	LNAME	;last char of string
230F	3E	02		1570	MVI	A,2	;constant of type 2 (char str)
2311	B7			1580	ORA	A	
2312	23			1590	INX	H	
2313	23			1600	INX	H	
2314	22	5C	20	1610	SHLD	CURSOR	
2317	C9			1620	RET		
2318	FE	27		1630	CPI	27H	;test for prime
231A	C2	3C	23	1640	JNZ	CN9	
231D	23			1650	INX	H	
231E	22	56	20	1660	SHLD	FNAME	
2321	7E			1670	MOV	A,M	;scan for matching prime
2322	FE	27		1680	CPI	27H	
2324	CA	34	23	1690	JZ	CN11	
2327	23			1700	INX	H	
2328	EB			1710	XCHG		;cursor check
2329	2A	60	20	1720	LHLD	PROGEND	
232C	19			1730	DAD	D	
232D	EB			1740	XCHG		
232E	D2	21	23	1750	JNC	CN12	
2331	C3	3E	23	1760	JMP	CNERR	
2334	3E	03		1770	MVI	A,3	;found matching prime
2336	B7			1780	ORA	A	
2337	23			1790	INX	H	
2338	22	5C	20	1800	SHLD	CURSOR	
233B	C9			1810	RET		
233C	AF			1820	XRA	A	;no match
233D	C9			1830	RET		
233E	CD	F1	21	1840	CNERR	CALL	ESET
2341	02			1850	DB	CURSERR	
2342	C9			1860	RET		
2343				1870			
2343				1880			
2343	11	9F	20	1890	REM	LXI	D,NEWLINE

;skips over remarks and/or end-of-lines in any order.

tiny-c/8080 Version 80-01-01

2346	CD	1F	22	1900	CALL	LIT	
2349	C2	43	23	1910	JNZ	REM	
234C	11	A1	20	1920	LXI	D,CMNT	
234F	CD	1F	22	1930	CALL	LIT	
2352	C8			1940	RZ		
2353	06	01		1950	MVI	B,1	;can't match anything
2355	0E	0D		1960	MVI	C,ASCRT	
2357	CD	54	22	1970	CALL	SKIP	
235A	D8			1980	RC		;error check
235B	C3	43	23	1990	JMP	REM	
235E				2000			
235E				2010			;HL points to start of digit string. Converts to integer
235E				2020			; leaving result in DE. Uses all digits, even if DE
235E				2030			; overflows. First nondigit stops scan.
235E	EB			2040	ATON	XCHG	;pointer into DE
235F	21	00	00	2050		LXI	H,0 ;answer developed here
2362	1A			2060	AN2	LDAX	D ;next ascii
2363	D6	30		2070		SUI	48
2365	DA	7B	23	2080		JC	AN3 ;test for digit
2368	FE	0A		2090		CPI	10
236A	D2	7B	23	2100		JNC	AN3
236D	44			2110		MOV	B,H ;digit, set HL=10*HL+A
236E	4D			2120		MOV	C,L
236F	29			2130		DAD	H
2370	29			2140		DAD	H
2371	09			2150		DAD	B
2372	29			2160		DAD	H
2373	4F			2170		MOV	C,A
2374	06	00		2180		MVI	B,0
2376	09			2190		DAD	B
2377	13			2200		INX	D ;bump pointer
2378	C3	62	23	2210		JMP	AN2
237B	EB			2220	AN3	XCHG	;answer -> DE
237C	C9			2230		RET	
237D				2240			
237D				2250			;HL points to beginning of ascii integer, possibly
237D				2260			; signed. Converts to integer and leaves value in DE.
237D	00			2270	AISGN	DB	0 ;nonzero for -

237E AF	2280 ATOI	XRA	A	
237F 32 7D 23	2290	STA	AISGN	
2382 7E	2300 AI6	MOV	A,M	;skip blanks
2383 FE 20	2310	CPI	' '	
2385 C2 8C 23	2320	JNZ	AI2	
2388 23	2330	INX	H	
2389 C3 82 23	2340	JMP	AI6	
238C FE 2D	2350 AI2	CPI	'-'	;test sign
238E C2 95 23	2360	JNZ	AI3	
2391 32 7D 23	2370	STA	AISGN	;is -
2394 23	2380	INX	H	
2395 FE 2B	2390 AI3	CPI	'+'	
2397 C2 9B 23	2400	JNZ	AI4	
239A 23	2410	INX	H	
239B 7E	2420 AI4	MOV	A,M	;skip more blanks
239C FE 20	2430	CPI	' '	
239E C2 A5 23	2440	JNZ	AI5	
23A1 23	2450	INX	H	
23A2 C3 9B 23	2460	JMP	AI4	
23A5 CD 5E 23	2470 AI5	CALL	ATON	;does the digits
23A8 3A 7D 23	2480	LDA	AISGN	;magnitude in DE
23AB B7	2490	ORA	A	
23AC C8	2500	RZ		
23AD C3 74 21	2510	JMP	DENEG	;computes negative and returns
23B0	8888 END3	EQU	\$	

tiny-c/8080 Version 80-01-01

```
23B0      0000      ;
23B0      0010      ;@@@@@@@@@ SYMBOL TOOLS @@@@@@@@@@@@
23B0      0020      ;
23B0      0030      ;allocate reference in FUNB for variables of a function
23B0 2A 52 20      0040 NEWFUN LHL D CURFUN
23B3 11 06 00      0050 LXI D,6 ;bump CURFUN by 6
23B6 19            0060 DAD D
23B7 22 52 20      0070 SHLD CURFUN
23BA EB            0080 XCHG ;test too many active functions
23BB 2A 3C 20      0090 LHL EFUN
23BE 19            0100 DAD D
23BF EB            0110 XCHG
23C0 D2 C8 23      0120 JNC NF2
23C3 CD F1 21      0130 CALL ESET
23C6 11            0140 DB TMFUERR
23C7 C9            0150 RET
23C8 3A 50 20      0160 NF2 LDA NXTVAR ;init first and last var
23CB 77            0170 MOV M,A ;fv lo byte
23CC D6 0E         0180 SUI 6+VLEN
23CE 4F            0190 MOV C,A ;lv lo byte -> C for now
23CF 3A 51 20      0200 LDA NXTVAR+1
23D2 23            0210 INX H
23D3 77            0220 MOV M,A ;fv hi byte
23D4 DE 00         0230 SBI 0 ;picks up possible carry
23D6 23            0240 INX H
23D7 71            0250 MOV M,C ;lv lo byte
23D8 23            0260 INX H
23D9 77            0270 MOV M,A ;lv hi byte
23DA 3A 5E 20      0280 LDA PRUSED ;now set up backup pointer
23DD 23            0290 INX H
23DE 77            0300 MOV M,A ;bu lo byte
23DF 3A 5F 20      0310 LDA PRUSED+1
23E2 23            0320 INX H
23E3 77            0330 MOV M,A ;bu hi bytv
23E4 C9            0340 RET ;all done
23E5            0350 ;
23E5            0360 ;deallocate variables of last function.
23E5 2A 52 20      0370 FUNDONE LHL CURFUN
```


23E8	7E		0380	MOV	A,M	
23E9	32	50 20	0390	STA	NXTVAR	;lo byte
23EC	23		0400	INX	H	
23ED	7E		0410	MOV	A,M	
23EE	32	51 20	0420	STA	NXTVAR+1	
23F1	23		0430	INX	H	
23F2	23		0440	INX	H	
23F3	23		0450	INX	H	
23F4	7E		0460	MOV	A,M	
23F5	32	5E 20	0470	STA	PRUSED	
23F8	23		0480	INX	H	
23F9	7E		0490	MOV	A,M	
23FA	32	5F 20	0500	STA	PRUSED+1	
23FD	11	F5 FF	0510	LXI	D,-11	
2400	19		0520	DAD	D	;subtract 5 for above INX's,
2401	22	52 20	0530	SHLD	CURFUN	; plus 5 more to pop FUNB.
2404	C9		0540	RET		
2405			0550			
2405			0560			;allocate a variable. Class in A, size in B, len in DE,
2405			0570			; passed value in HL.
2405	00		0580	CLASS	DB	0 ;temps used by newvar
2406	00		0590	OBSIZE	DB	0
2407	00 00		0600	PASSED	DW	0
2409	00 00		0610	LEN	DW	0
240B	00 00		0620	FVAL	DW	0
240D	00 00		0630	KF	DW	0
240F			0640			
240F	32	05 24	0650	NEWVAR	STA	CLASS
2412	78		0660	MOV	A,B	
2413	32	06 24	0670	STA	OBSIZE	
2416	22	07 24	0680	SHLD	PASSED	
2419	EB		0690	XCHG		
241A	22	09 24	0700	SHLD	LEN	
241D	2A	50 20	0710	LHLD	NXTVAR	
2420	CD	6B 25	0720	CALL	CANON	;put canonical form of name
2423			0730			into (NXTVAR). Leaves HL
2423			0740			; pointing to last byte of NAME of VARB.
2423	23		0750	INX	H	; -> CLASS in VARB.

tiny-c/8080 Version 80-01-01

2424 3A 05 24	0760	LDA	CLASS	
2427 77	0770	MOV	M,A	
2428 23	0780	INX	H	;-> OBJSIZE in VARB.
2429 3A 06 24	0790	LDA	OBJSIZE	
242C 77	0800	MOV	M,A	
242D 23	0810	INX	H	;-> LEN in VARB (2 bytes).
242E 3A 09 24	0820	LDA	LEN	
2431 77	0830	MOV	M,A	
2432 23	0840	INX	H	
2433 3A 0A 24	0850	LDA	LEN+1	
2436 77	0860	MOV	M,A	
2437 23	0870	INX	H	
2438 22 0B 24	0880	SHLD	FVAL	;address where fval will be put
243B 3A 05 24	0890	LDA	CLASS	
243E B7	0900	ORA	A	;if class is 0, or not a passed
243F CA 4A 24	0910	JZ	NR2	; arg, then get value space.
2442 2A 07 24	0920	LHLD	PASSED	
2445 7D	0930	MOV	A,L	
2446 B4	0940	ORA	H	
2447 C2 86 24	0950	JNZ	NR3	
244A 2A 5E 20	0960	LHLD	PRUSED	;get value space
244D 23	0970	INX	H	; starting at PRUSED + 1
244E 22 0D 24	0980	SHLD	KF	;Put in KF for later use.
2451 EB	0990	XCHG		
2452 2A 0B 24	1000	LHLD	FVAL	
2455 73	1010	MOV	M,E	
2456 23	1020	INX	H	
2457 72	1030	MOV	M,D	;fval part of varb set to
2458 2A 09 24	1040	LHLD	LEN	; prused+1. Now bump prused
245B EB	1050	XCHG		; by obsize*len.
245C 2A 5E 20	1060	LHLD	PRUSED	
245F 3A 06 24	1070	LDA	OBJSIZE	
2462 19	1080	DAD	D	
2463 3D	1090	DCR	A	
2464 CA 68 24	1100	JZ	\$+1	
2467 19	1110	DAD	D	
2468 22 5E 20	1120	SHLD	PRUSED	
246B EB	1130	XCHG		;test if allocation exceeds

246C	2A	44	20	1140	LHLD	EPR	; limits of prog space.
246F	19			1150	DAD	D	
2470	EB			1160	XCHG		
2471	D2	79	24	1170	JNC	NR4	
2474	CD	F1	21	1180	CALL	ESET	;RAM exceeded
2477	13			1190	DB	TMVLERR	
2478	C9			1200	RET		
2479	2A	0D	24	1210	LHLD	KF	;zero the allocated space
247C	EB			1220	XCHG		
247D	2A	5E	20	1230	LHLD	PRUSED	
2480	CD	09	22	1240	CALL	ZERO	
2483	C3	95	24	1250	JMP	NR5	;end of space allocation
2486	2A	0B	24	1260	LHLD	FVAL	;Value is passed and is a
2489	3A	07	24	1270	LDA	PASSED	; class > 0. Put value in fval
248C	77			1280	MOV	M,A	; part of VARB. Dont allocate
248D	23			1290	INX	H	; space.
248E	3A	08	24	1300	LDA	PASSED+1	
2491	77			1310	MOV	M,A	
2492	C3	AB	24	1320	JMP	NR6	
2495	3A	05	24	1330	LDA	CLASS	;if passed & class is 0 move
2498	B7			1340	ORA	A	; the passed value into the
2499	C2	AB	24	1350	JNZ	NR6	; allocated space.
249C	2A	07	24	1360	LHLD	PASSED	
249F	7C			1370	MOV	A,H	
24A0	B5			1380	ORA	L	
24A1	CA	AB	24	1390	JZ	NR6	
24A4	EB			1400	XCHG		;passed -> DE
24A5	2A	0D	24	1410	LHLD	KF	
24A8	73			1420	MOV	M,E	;lo byte of passed value
24A9	23			1430	INX	H	
24AA	72			1440	MOV	M,D	;hi byte, or junk if only one
24AB				1450			byte passed. Who cares.
24AB	2A	52	20	1460	LHLD	CURFUN	;in FUNB set lvar part to this
24AE	23			1470	INX	H	; variable.
24AF	23			1480	INX	H	
24B0	3A	50	20	1490	LDA	NXTVAR	
24B3	77			1500	MOV	M,A	
24B4	23			1510	INX	H	

tiny-c/8080 Version 80-01-01

24B5	3A	51	20	1520	LDA	NXTVAR+1	
24B8	77			1530	MOV	M,A	
24B9	2A	50	20	1540	LHLD	NXTVAR	;increment NXTVAR
24BC	11	0E	00	1550	LXI	D,6+VLEN	; by 6 + vlen
24BF	19			1560	DAD	D	
24C0	22	50	20	1570	SHLD	NXTVAR	
24C3	EB			1580	XCHG		;test if too many variables
24C4	2A	40	20	1590	LHLD	EVAR	
24C7	19			1600	DAD	D	
24C8	EB			1610	XCHG		
24C9	2A	0B	24	1620	LHLD	FVAL	
24CC	D0			1630	RNC		;normal return, FVAL in HL.
24CD	CD	F1	21	1640	CALL	ESET	;VARB exceeded.
24D0	12			1650	DB	TMVRERR	
24D1	C9			1660	RET		
24D2				1670			
24D2				1680			;ADDRVAL looks up a symbol pointed to by FNAME,LNAME.
24D2				1690			; Returns address in HL, class in A, size in B, and
24D2				1700			; length in DE. Sets err if symbol cannot be found.
24D2				1710			; Searches 3 areas:
24D2				1720		area 0	locals
24D2				1730		1	globals
24D2				1740		2	library symbols
24D2				1750	NAME DS	VLEN	;holds canonical form of name
24DA	00	00		1760	PVAR DW	0	
24DC	00			1770	AREA DB	0	
24DD	00	00		1780	SFUN DW	0	
24DF	00	00		1790	LAST DW	0	
24E1				1800			
24E1	2A	52	20	1810	ADDRVAL LHLD	CURFUN	
24E4	22	DD	24	1820	SHLD	SFUN	;search locals first
24E7	21	D2	24	1830	LXI	H,NAME	
24EA	CD	6B	25	1840	CALL	CANON	
24ED	AF			1850	XRA	A	
24EE	32	DC	24	1860	STA	AREA	;area 0
24F1	2A	DD	24	1870	AD8 LHLD	SFUN	;variable search area
24F4	5E			1880	MOV	E,M	
24F5	23			1890	INX	H	

24F6	56		1900	MOV	D,M	;fvar of search area -> DE
24F7	23		1910	INX	H	
24F8	4E		1920	MOV	C,M	
24F9	23		1930	INX	H	
24FA	46		1940	MOV	B,M	;lvar -> BC
24FB	EB		1950	XCHG		
24FC	22	DA 24	1960	SHLD	PVAR	;currently searched variable
24FF	60		1970	MOV	H,B	
2500	69		1980	MOV	L,C	
2501	22	DF 24	1990	SHLD	LAST	;last to search in this area
2504	2A	DA 24	2000	LHLD	PVAR	;begin search loop
2507	3A	DF 24	2010	LDA	LAST	;test for end of loop
250A	95		2020	SUB	L	
250B	3A	E0 24	2030	LDA	LAST+1	
250E	9C		2040	SBB	H	
250F	DA	45 25	2050	JC	AD3	
2512	0E	08	2060	MVI	C,VLEN	;number of chars to match
2514	11	D2 24	2070	LXI	D,NAME	;match string address
2517	1A		2080	LDAX	D	; (HL already as table entry)
2518	BE		2090	CMP	M	
2519	C2	38 25	2100	JNZ	AD5	;no match
251C	0D		2110	DCR	C	
251D	13		2120	INX	D	
251E	23		2130	INX	H	
251F	C2	17 25	2140	JNZ	AD4	;next char
2522	7E		2150	MOV	A,M	;MATCH. HL points to class.
2523	23		2160	INX	H	
2524	46		2170	MOV	B,M	;obsize
2525	23		2180	INX	H	
2526	5E		2190	MOV	E,M	
2527	23		2200	INX	H	
2528	56		2210	MOV	D,M	;length
2529	23		2220	INX	H	
252A	B7		2230	ORA	A	;if class > 0 & class < 'E'
252B	CA	31 25	2240	JZ	AD9	; then return address of fval
252E	FE	45	2250	CPI	'E'	; part of VARB, which is alrdy
2530	C0		2260	RNZ		; in HL.
2531	D5		2270	PUSH	D	;otherwise return contents of

tiny-c/8080 Version 80-01-01

2532 5E	2280	MOV	E,M	; fval part of VARB.
2533 23	2290	INX	H	
2534 56	2300	MOV	D,M	
2535 EB	2310	XCHG		
2536 D1	2320	POP	D	
2537 C9	2330	RET		
2538 2A DA 24	2340	LHLD	PVAR	;go to next variable
253B 11 0E 00	2350	LXI	D,VLEN+6	
253E 19	2360	DAD	D	
253F 22 DA 24	2370	SHLD	PVAR	
2542 C3 07 25	2380	JMP	AD2	
2545 3A DC 24	2390	LDA	AREA	;go to next area
2548 B7	2400	ORA	A	
2549 C2 59 25	2410	JNZ	AD6	
254C 2A 54 20	2420	LHLD	CURGLBL	;second search area, globals
254F 22 DD 24	2430	SHLD	SFUN	
2552 3C	2440	INR	A	
2553 32 DC 24	2450	STA	AREA	
2556 C3 F1 24	2460	JMP	AD8	
2559 FE 02	2470	CPI	2	
255B F2 64 25	2480	JP	ADERR	
255E 2A 3A 20	2490	LHLD	BFUN	;third area is library, which
2561 C3 4F 25	2500	JMP	AD7	; is at beginning of FUNB.
2564 CD F1 21	2510	CALL	ESET	
2567 1A	2520	DB	SYMERRA	
2568 C9	2530	RET		
2569	2540			
2569	2550			;canonicalizes symbol from FNAME to LNAME inclusive,
2569	2560			; putting form with VLEN chars in (HL).
2569 00 00	2570	OUTNAME DW	0	
256B 22 69 25	2580	CANON SHLD	OUTNAME	
256E 3E 08	2590	MVI	A,VLEN	;zero output field
2570 06 00	2600	MVI	B,0	
2572 48	2610	MOV	C,B	;zero C for later
2573 70	2620	MOV	M,B	
2574 3D	2630	DCR	A	
2575 CA 7C 25	2640	JZ	CA3	
2578 23	2650	INX	H	

2579 C3 73 25	2660		JMP	CA2	
257C E5	2670	CA3	PUSH	H	;save pointer to last byte
257D 2A 56 20	2680		LHLD	FNAME	;compute symbols actual length
2580 3A 58 20	2690		LDA	LNAME	
2583 95	2700		SUB	L	
2584 3C	2710		INR	A	
2585 FE 08	2720		CPI	VLEN	
2587 FA 8D 25	2730		JM	\$+3	
258A 3E 08	2740		MVI	A,VLEN	;A now has number of chars to
258C 4F	2750		MOV	C,A	; be moved, and C is nonzero
258D EB	2760		XCHG		; iff act len > VLEN.
258E 47	2770		MOV	B,A	
258F 2A 69 25	2780		LHLD	OUTNAME	;FNAME -> DE, OUTNAME -> HL
2592 1A	2790	CA4	LDAX	D	;copy loop
2593 77	2800		MOV	M,A	
2594 05	2810		DCR	B	
2595 CA 9D 25	2820		JZ	CA5	
2598 13	2830		INX	D	
2599 23	2840		INX	H	
259A C3 92 25	2850		JMP	CA4	
259D E1	2860	CA5	POP	H	;pointer to last byte
259E AF	2870		XRA	A	
259F B1	2880		ORA	C	;test if short name
25A0 C8	2890		RZ		
25A1 EB	2900		XCHG		;long name, put last char in
25A2 2A 58 20	2910		LHLD	LNAME	; the canon form.
25A5 7E	2920		MOV	A,M	;last char of name
25A6 EB	2930		XCHG		
25A7 77	2940		MOV	M,A	;into last pos of outname
25A8 C9	2950		RET		
25A9	8888	END4	EQU	\$	

25A9	0000	;ASGN is the expression evaluator,so called because
25A9	0010	; the highest form of an expression is an assignment.
25A9	0020	; An asgn is a reln or an lvalue = asgn. Note that
25A9	0030	; reln can match an lvalue.
25A9	0040	;Returns non-zero if valid expression, 0 if invalid.
25A9 CD CA 25	0050	ASGN CALL RELN ;stacked as lvalue if that's
25AC	0060	; what it is.
25AC 11 B6 20	0070	LXI D,XEQ ; test for =
25AF CD 1F 22	0080	CALL LIT
25B2 CA BF 25	0090	JZ A2
25B5 CD A9 25	0100	CALL ASGN
25B8 3A 48 20	0110	LDA ERR ;check for error
25BB B7	0120	ORA A
25BC CC C0 20	0130	CZ EQ ;perform assignment
25BF 3A 48 20	0140	A2 LDA ERR ;return 0 (i.e. no match) if
25C2 B7	0150	ORA A ; there was an error
25C3 CA C8 25	0160	JZ A3
25C6 AF	0170	XRA A
25C7 C9	0180	RET
25C8 3D	0190	A3 DCR A ;no error so return non-zero A
25C9 C9	0200	RET
25CA	0210	;
25CA	0220	;a RELN is an expr or a comparison of exprs
25CA CD 52 26	0230	RELN CALL EXPR
25CD 11 BB 20	0240	LXI D,LE ; <=
25D0 CD 1F 22	0250	CALL LIT
25D3 CA E5 25	0260	JZ R2
25D6 CD 52 26	0270	CALL EXPR ;right side
25D9 CD EF 20	0280	CALL TOPDIF ;sets Z,C flags. C set as
25DC CA BF 21	0290	JZ PONE ; though it were S. Must be
25DF DA BF 21	0300	JC PONE ; zero or negative for true.
25E2 C3 C5 21	0310	JMP PZERO ;These jumps all call/rets.
25E5 11 B8 20	0320	R2 LXI D,GE ; >=
25E8 CD 1F 22	0330	CALL LIT
25EB CA FD 25	0340	JZ R3
25EE CD 52 26	0350	CALL EXPR
25F1 CD EF 20	0360	CALL TOPDIF
25F4 CA BF 21	0370	JZ PONE

25F7 D2 BF 21	0380		JNC	PONE	
25FA C3 C5 21	0390		JMP	PZERO	
25FD 11 B5 20	0400	R3	LXI	D,EQEQ	; ==
2600 CD 1F 22	0410		CALL	LIT	
2603 CA 12 26	0420		JZ	R4	
2606 CD 52 26	0430		CALL	EXPR	
2609 CD EF 20	0440		CALL	TOPDIF	
260C CA BF 21	0450		JZ	PONE	
260F C3 C5 21	0460		JMP	PZERO	
2612 11 B2 20	0470	R4	LXI	D,NOTEQ	; !=
2615 CD 1F 22	0480		CALL	LIT	
2618 CA 27 26	0490		JZ	R5	
261B CD 52 26	0500		CALL	EXPR	
261E CD EF 20	0510		CALL	TOPDIF	
2621 C2 BF 21	0520		JNZ	PONE	
2624 C3 C5 21	0530		JMP	PZERO	
2627 11 B0 20	0540	R5	LXI	D,GT	; >
262A CD 1F 22	0550		CALL	LIT	
262D CA 3F 26	0560		JZ	R6	
2630 CD 52 26	0570		CALL	EXPR	
2633 CD EF 20	0580		CALL	TOPDIF	
2636 CA C5 21	0590		JZ	PZERO	
2639 DA C5 21	0600		JC	PZERO	
263C C3 BF 21	0610		JMP	PONE	
263F 11 AE 20	0620	R6	LXI	D,LT	; <
2642 CD 1F 22	0630		CALL	LIT	
2645 C8	0640		RZ		; no relational operator
2646 CD 52 26	0650		CALL	EXPR	
2649 CD EF 20	0660		CALL	TOPDIF	
264C DA BF 21	0670		JC	PONE	
264F C3 C5 21	0680		JMP	PZERO	
2652	0690				
2652	0700				
2652 11 AC 20	0710				
2655 CD 1F 22	0720				
2658 CA 6E 26	0730				
265B CD AA 26	0740				
265E CD 83 21	0750				

;an EXPR is a term or sum (diff) of terms.
 EXPR LXI D,XMINUS ; unary -
 CALL LIT
 JZ EX2
 CALL TERM
 CALL TOPTOI ;push negative of top back onto

tiny-c/8080 Version 80-01-01

2661	7B		0760	MOV	A,E	
2662	2F		0770	CMA		
2663	5F		0780	MOV	E,A	
2664	7A		0790	MOV	A,D	
2665	2F		0800	CMA		
2666	57		0810	MOV	D,A	
2667	13		0820	INX	D	
2668	CD C8	21	0830	CALL	PUSHK	
266B	C3 77	26	0840	JMP	EX3	
266E	11 AA	20	0850	EX2 LXI	D,XPLUS ;optional unary +	
2671	CD 1F	22	0860	CALL	LIT	
2674	CD AA	26	0870	CALL	TERM	
2677			0880		;first term is now stacked. Check for error so far.	
2677	3A 48	20	0890	EX3 LDA	ERR	
267A	B7		0900	ORA	A	
267B	C0		0910	RNZ		
267C	11 AA	20	0920	LXI	D,XPLUS ; +	
267F	CD 1F	22	0930	CALL	LIT	
2682	CA 94	26	0940	JZ	EX4	
2685	CD AA	26	0950	CALL	TERM	
2688	CD A0	21	0960	CALL	POPTWO ;top two values on stack are	
268B			0970		actualized and put into	
268B			0980		(BC) and (DE).	
268B	CD FC	20	0990	CALL	DADD ; (BC)+(DE)->(DE)	
268E	CD C8	21	1000	CALL	PUSHK ; sum onto stack.	
2691	C3 77	26	1010	JMP	EX3 ;back for more terms	
2694	11 AC	20	1020	EX4 LXI	D,XMINUS ; -	
2697	CD 1F	22	1030	CALL	LIT	
269A	C8		1040	RZ	;no more terms	
269B	CD AA	26	1050	CALL	TERM	
269E	CD A0	21	1060	CALL	POPTWO	
26A1	CD F2	20	1070	CALL	DSUB	
26A4	CD C8	21	1080	CALL	PUSHK	
26A7	C3 77	26	1090	JMP	EX3 ;back for more terms.	
26AA			1100			
26AA			1110		;a term is a factor or a product of factors.	
26AA	CD FB	26	1120	TERM CALL	FACTOR	
26AD	3A 48	20	1130	TE2 LDA	ERR ;check for error so far	

26B0 B7	1140	ORA	A
26B1 C0	1150	RNZ	
26B2 11 A2 20	1160	LXI	D,XSTAR ; *
26B5 CD 1F 22	1170	CALL	LIT
26B8 CA CA 26	1180	JZ	TE3
26BB CD FB 26	1190	CALL	FACTOR
26BE CD A0 21	1200	CALL	POPTWO
26C1 CD 06 21	1210	CALL	DMPY
26C4 CD C8 21	1220	CALL	PUSHK
26C7 C3 AD 26	1230	JMP	TE2 ;back for more factors.
26CA CD 43 23	1240	CALL	REM ;make sure no /*
26CD 11 A8 20	1250	LXI	D,XSLASH ; /
26D0 CD 1F 22	1260	CALL	LIT
26D3 CA E5 26	1270	JZ	TE4
26D6 CD FB 26	1280	CALL	FACTOR
26D9 CD A0 21	1290	CALL	POPTWO
26DC CD 6F 21	1300	CALL	DDIV
26DF CD C8 21	1310	CALL	PUSHK
26E2 C3 AD 26	1320	JMP	TE2
26E5 11 A6 20	1330	LXI	D,XPCNT ; %
26E8 CD 1F 22	1340	CALL	LIT
26EB C8	1350	RZ	;no more factors.
26EC CD FB 26	1360	CALL	FACTOR
26EF CD A0 21	1370	CALL	POPTWO
26F2 CD 31 21	1380	CALL	DREM
26F5 CD C8 21	1390	CALL	PUSHK
26F8 C3 AD 26	1400	JMP	TE2
26FB	1410		
26FB	1420		;a FACTOR is a (asgn), or a constant, or a variable
26FB	1430		; reference, or a function reference.
26FB 11 99 20	1440	FACTOR LXI	D,LPAR ; (
26FE CD 1F 22	1450	CALL	LIT
2701 CA 13 27	1460	JZ	FA2
2704 CD A9 25	1470	CALL	ASGN
2707 11 9B 20	1480	LXI	D,RPAR ;)
270A CD 1F 22	1490	CALL	LIT
270D C0	1500	RNZ	
270E CD F1 21	1510	CALL	ESET ;right paren error

tiny-c/8080 Version 80-01-01

2711 05	1520		DB	RPARERR	
2712 C9	1530		RET		
2713 CD B9 22	1540	FA2	CALL	CONST	;recognizes 3 types of constant
2716 CA 45 27	1550		JZ	FA5	; setting A accordingly.
2719 FE 01	1560		CPI	1	
271B C2 27 27	1570		JNZ	FA3	
271E 2A 56 20	1580		LHLD	FNAME	;type 1: integer. FNAME points
2721 CD 7E 23	1590		CALL	ATOI	; to beginning. ATOI converts
2724 C3 C8 21	1600		JMP	PUSHK	; it, leaving value in (DE).
2727 FE 02	1610	FA3	CPI	2	
2729 C2 39 27	1620		JNZ	FA4	
272C 3E 01	1630		MVI	A,1	;type 2: char string. Push
272E 06 41	1640		MVI	B,'A'	; class=1, lval='A', size=1,
2730 0E 01	1650		MVI	C,1	; and stuff=address of
2732 2A 56 20	1660		LHLD	FNAME	; beginning of string.
2735 EB	1670		XCHG		
2736 C3 CD 21	1680		JMP	PUSH	
2739 AF	1690	FA4	XRA	A	;type 3: char constant. Push
273A 06 41	1700		MVI	B,'A'	; class=0, lval='A', size=1,
273C 0E 01	1710		MVI	C,1	; and stuff=actual character.
273E 2A 56 20	1720		LHLD	FNAME	
2741 5E	1730		MOV	E,M	
2742 C3 CD 21	1740		JMP	PUSH	
2745 CD 9A 22	1750	FA5	CALL	SYMNAME	;not a constant, try symbol.
2748 CA 23 28	1760		JZ	FA6	
274B 2A 56 20	1770		LHLD	FNAME	;symbol. Test for special
274E 23	1780		INX	H	; symbol MC. First is symbol
274F 3A 58 20	1790		LDA	LNAME	; length exactly 2.
2752 BD	1800		CMP	L	
2753 C2 70 27	1810		JNZ	FA7	
2756 3A 59 20	1820		LDA	LNAME+1	
2759 BC	1830		CMP	H	
275A C2 70 27	1840		JNZ	FA7	
275D 7E	1850		MOV	A,M	;length is 2, and (HL)=FNAME.
275E FE 43	1860		CPI	'C'	
2760 C2 70 27	1870		JNZ	FA7	
2763 2B	1880		DCX	H	
2764 7E	1890		MOV	A,M	

2765 FE 4D	1900	CPI	'M'	
2767 C2 70 27	1910	JNZ	FA7	
276A 21 00 00	1920	LXI	H,0	
276D C3 80 2A	1930	JMP	ENTER	;causes machine call.
2770 CD E1 24	1940	CALL	ADDRVAL	;not MC, look up symbol.
2773 22 28 28	1950	SHLD	WHERE	
2776 32 05 24	1960	STA	CLASS	
2779 78	1970	MOV	A,B	;save results of lookup.
277A 32 06 24	1980	STA	OBSIZE	
277D EB	1990	XCHG		
277E 22 09 24	2000	SHLD	LEN	
2781 7A	2010	MOV	A,D	;where is now in DE
2782 B7	2020	ORA	A,E	
2783 CA 1E 28	2030	JZ	FA8	
2786 3A 05 24	2040	LDA	CLASS	
2789 FE 45	2050	CPI	'E'	;class E => function entry
278B CA 18 28	2060	JZ	FA9	
278E 11 99 20	2070	LXI	D,LPAR	;variable. Test for subscript.
2791 CD 1F 22	2080	CALL	LIT	
2794 CA 08 28	2090	JZ	FA10	
2797 3A 05 24	2100	LDA	CLASS	;subscripted, class must be > 0
279A 3D	2110	DCR	A	
279B 32 05 24	2120	STA	CLASS	;class of element is one less
279E F2 A6 27	2130	JP	FA11	; than class of array.
27A1 CD F1 21	2140	CALL	ESET	
27A4 07	2150	DB	CLASERR	
27A5 C9	2160	RET		
27A6 2A 28 28	2170	LHLD	WHERE	;replace where by two bytes
27A9 5E	2180	MOV	E,M	; referenced by where.
27AA 23	2190	INX	H	
27AB 56	2200	MOV	D,M	
27AC D5	2210	PUSH	D	;save where, len, class,
27AD 2A 09 24	2220	LHLD	LEN	; obsize.
27B0 E5	2230	PUSH	H	
27B1 2A 05 24	2240	LHLD	CLASS	; (also gets obsize)
27B4 E5	2250	PUSH	H	
27B5 CD A9 25	2260	CALL	ASGN	;evaluate subscript
27B8 E1	2270	POP	H	

tiny-c/8080 Version 80-01-01

27B9	22 05 24	2280	SHLD	CLASS	;restore everything
27BC	E1	2290	POP	H	
27BD	22 09 24	2300	SHLD	LEN	
27C0	E1	2310	POP	H	
27C1	22 28 28	2320	SHLD	WHERE	
27C4	C8	2330	RZ		;assign error
27C5	11 9B 20	2340	LXI	D, RPAR	;skip)
27C8	CD 1F 22	2350	CALL	LIT	
27CB	CD 83 21	2360	CALL	TOPTOI	;subscript value -> DE
27CE	EB	2370	XCHG		
27CF	22 2A 28	2380	SHLD	SUBSCR	
27D2	EB	2390	XCHG		
27D3	2A 09 24	2400	LHLD	LEN	
27D6	7D	2410	MOV	A, L	
27D7	3D	2420	DCR	A	
27D8	B4	2430	ORA	H	;for LEN = 1 skip subscript
27D9	CA F3 27	2440	JZ	FA12	; check.
27DC	3A 05 24	2450	LDA	CLASS	
27DF	B7	2460	ORA	A	
27E0	C2 F3 27	2470	JNZ	FA12	;skip for pointers, too.
27E3	B2	2480	ORA	D	
27E4	FA EF 27	2490	JM	SUBERR	;cant be negative
27E7	44	2500	MOV	B, H	;len -> BC
27E8	4D	2510	MOV	C, L	
27E9	CD F2 20	2520	CALL	DSUB	
27EC	DA F3 27	2530	JC	FA12	;subscr-len must be negative
27EF	CD F1 21	2540	CALL	ESET	
27F2	06	2550	DB	RANGERR	
27F3	2A 2A 28	2560	LHLD	SUBSCR	
27F6	EB	2570	XCHG		;where =+ subscr * obsize
27F7	2A 28 28	2580	LHLD	WHERE	
27FA	3A 06 24	2590	LDA	OBSIZE	
27FD	3D	2600	DCR	A	
27FE	FA 05 28	2610	JM	FA14	
2801	19	2620	DAD	D	
2802	C3 FD 27	2630	JMP	FA13	
2805	22 28 28	2640	SHLD	WHERE	
2808	3A 06 24	2650	LDA	OBSIZE	;push class, 'L', obsize,

280B 4F	2660	MOV	C,A	; stuff=where.
280C 3A 05 24	2670	LDA	CLASS	
280F 06 4C	2680	MVI	B,'L'	
2811 2A 28 28	2690	LHLD	WHERE	
2814 EB	2700	XCHG		
2815 C3 CD 21	2710	JMP	PUSH	;call/ret
2818 2A 28 28	2720	LHLD	WHERE	
281B C3 80 2A	2730	JMP	ENTER	;call/ret
281E CD F1 21	2740	CALL	ESET	;symbol error
2821 03	2750	DB	SYMMERR	
2822 C9	2760	RET		
2823 CD F1 21	2770	CALL	ESET	;cannot recognize factor
2826 09	2780	DB	SYNXERR	
2827 C9	2790	RET		
2828	2800			
2828	2810			
2828 00 00	2820	WHERE	DW	0
282A 00 00	2830	SUBSCR	DW	0
282C	8888	END5	EQU	\$

;locals used by ASGN, etc.

tiny-c/8080 Version 80-01-01

282C		0000		;SKIPST skips over a (possibly compound) statement,
282C		0010		; including whole nested sets of if-then-elses.
282C		0020		; Assumes balanced [], even within comments.
282C	CD 43 23	0030	SKIPST CALL	REM
282F	11 95 20	0040	LXI	D,LB ;test for [
2832	CD 1F 22	0050	CALL	LIT
2835	CA 42 28	0060	JZ	SS2
2838	06 5B	0070	MVI	B,['['
283A	0E 5D	0080	MVI	C,']'
283C	CD 54 22	0090	CALL	SKIP
283F	C3 43 23	0095	JMP	REM ;and done
2842	11 63 20	0100	SS2 LXI	D,XIF ;test for if
2845	C3 C2 30	0110	JMP	XX3
2848	CA 67 28	0120	SS7 JZ	SS3
284B	11 99 20	0130	SS6 LXI	D,LPAR
284E	CD 1F 22	0140	CALL	LIT
2851	06 28	0150	MVI	B,'('
2853	0E 29	0160	MVI	C,')'
2855	CD 54 22	0170	CALL	SKIP ;skip over (condition) part
2858	CD 2C 28	0180	CALL	SKIPST ;skip then part
285B	11 66 20	0190	LXI	D,XELS ;test for ELSE
285E	CD 1F 22	0200	CALL	LIT
2861	C4 2C 28	0210	CNZ	SKIPST ;skip else part
2864	C3 43 23	0220	JMP	REM ;and done
2867	2A 5C 20	0230	SS3 LHL	D CURSOR ;simple statement, move cursor
286A	7E	0240	SS4 MOV	A,M ; past next ; or return.
286B	FE 0D	0242	CPI	ASCSET
286D	CA 82 28	0244	JZ	SS5
2870	FE 3B	0250	CPI	','
2872	CA 82 28	0260	JZ	SS5
2875	23	0270	INX	H
2876	EB	0280	XCHG	;test cursor overflow
2877	2A 60 20	0282	LHL	D PROGEND
287A	19	0284	DAD	D
287B	EB	0286	XCHG	
287C	D2 6A 28	0290	JNC	SS4
287F	C3 43 23	0300	JMP	REM ;and done
2882	23	0310	SS5 INX	H

2883 22 5C 20	0320	SHLD	CURSOR	
2886 C3 43 23	0330	JMP	REM	;and done
2889	0340			
2889	0350			;VALLOC parses one variable behind INT or CHAR and
2889	0360			; makes allocation and symbol entry.
2889 00	0370	TYPE	DB	0 ;'C' or 'I'
288A 00 00	0380	VPASSED	DW	0 ;0 for global or local, two
288C	0390			byte value if param to fnction
288C	0400			; It turns out a 0 valued parameter gets the same
288C	0410			treatment as a local.
288C 00	0420	VCLASS	DB	0 ;defined in globals section.
288D 00 00	0430	ALEN	DW	0 ;elements in an array.
288F	0440			
288F 32 89 28	0450	VALLOC	STA	TYPE
2892 22 8A 28	0460		SHLD	VPASSED
2895 CD 9A 22	0470		CALL	SYMNAME ;sets FNAME, LNAME around symb1
2898 CA F8 28	0480		JZ	V2 ;error if no symbol.
289B AF	0490		XRA	A
289C 32 8C 28	0500		STA	VCLASS ;assume class 0 (not an array)
289F 11 99 20	0510		LXI	D,LPAR
28A2 CD 1F 22	0520		CALL	LIT
28A5 CA DA 28	0530		JZ	V3
28A8 2A 56 20	0540		LHLD	FNAME ;array, evaluate subscript
28AB E5	0550		PUSH	H ; expression. Must push FNAME,
28AC 2A 58 20	0560		LHLD	LNAME ; LNAME, and class, because
28AF E5	0570		PUSH	H ; subscripts may invoke
28B0 3A 8C 28	0580		LDA	VCLASS ; functions which themselves
28B3 3C	0590		INR	A ; allocate variables.
28B4 F5	0600		PUSH	P
28B5 CD A9 25	0610		CALL	ASGN
28B8 F1	0620		POP	P ;restore pushed stuff.
28B9 32 8C 28	0630		STA	VCLASS
28BC E1	0640		POP	H
28BD 22 58 20	0650		SHLD	LNAME
28C0 E1	0660		POP	H
28C1 22 56 20	0670		SHLD	FNAME
28C4 3A 48 20	0680		LDA	ERR ;test for error in ASGN
28C7 B7	0690		ORA	A

tiny-c/8080 Version 80-01-01

28C8 C0	0700		RNZ	
28C9 11 9B 20	0710		LXI	D, RPAR
28CC CD 1F 22	0720		CALL	LIT ;skip)
28CF CD 83 21	0730		CALL	TOPTOI ;value of subscript + 1 into
28D2 13	0740		INX	D ; LEN
28D3 EB	0750		XCHG	
28D4 22 8D 28	0760		SHLD	ALEN
28D7 C3 E0 28	0770		JMP	V5
28DA 21 01 00	0780	V3	LXI	H,1 ;non-subscripted variable
28DD 22 8D 28	0790		SHLD	ALEN ; has ALEN 1.
28E0 3A 89 28	0800	V5	LDA	TYPE ;object size is 1 of 'C', 2 for
28E3 06 01	0810		MVI	B,1 ; 'I'
28E5 FE 43	0820		CPI	'C'
28E7 CA EB 28	0830		JZ	V7
28EA 04	0840		INR	B ;obsize in B
28EB 3A 8C 28	0850	V7	LDA	VCLASS ;class in A
28EE 2A 8D 28	0860		LHLD	ALEN ;len in DE.
28F1 EB	0870		XCHG	
28F2 2A 8A 28	0880		LHLD	VPASSED ;passed in HL
28F5 C3 0F 24	0890		JMP	NEWVAR ;call/ret, NEWVAR allocates the
28F8	0900			variable
28F8 CD F1 21	0910	V2	CALL	ESET
28FB 03	0920		DB	SYMERR
28FC C9	0930		RET	
28FD	8888	END6	EQU	\$

28FD		0000				
28FD		0010				; @@@@ tiny - c interpreter @@@@
28FD		0020				
28FD		0030				; ST interprets a possibly compound statement
28FD		0040				
28FD	CD 64 2A	0050	ST	CALL	QUIT	; test if program should quit.
2900	3A 48 20	0060		LDA	ERR	
2903	B7	0070		ORA	A	
2904	C0	0080		RNZ		
2905	CD 43 23	0090		CALL	REM	; pass over remarks and/or
2908		0100				end of line
2908	CD 25 20	0110		CALL	STBEGIN	; bugout for blips, statistics,
290B		0120				; etc, user provided.
290B	2A 5C 20	0130	ST2	LHLD	CURSOR	; capture cursor
290E	22 5A 20	0140		SHLD	STCURS	
2911	CD 25 2A	0150		CALL	DECL	; test for declaration
2914	C2 43 23	0160		JNZ	REM	
2917	11 95 20	0170		LXI	D, LB	; test for left bracket
291A	CD 1F 22	0180		CALL	LIT	
291D	CA 40 29	0190		JZ	TIF	
2920	CD 43 23	0200		CALL	REM	
2923	3A 48 20	0210	CMPND	LDA	ERR	; compound statement. Execute
2926	47	0220		MOV	B, A	; each of its inner stmts.
2927	3A 4C 20	0230		LDA	LEAVE	; Exit on error, leave, break,
292A	B0	0240		ORA	B	; or] literal.
292B	47	0250		MOV	B, A	
292C	3A 4D 20	0260		LDA	BRAKE	
292F	B0	0270		ORA	B	
2930	C0	0280		RNZ		
2931	11 97 20	0290		LXI	D, RB	;]
2934	CD 1F 22	0300		CALL	LIT	
2937	C2 43 23	0310		JNZ	REM	; and done
293A	CD FD 28	0320		CALL	ST	; recursive call to ST
293D	C3 23 29	0330		JMP	CMPND	; then do next statement.
2940	11 63 20	0340	TIF	LXI	D, XIF	; test for IF
2943	CD 1F 22	0350		CALL	LIT	
2946	CA 7B 29	0360		JZ	TWHI	
2949	11 99 20	0370		LXI	D, LPAR	; skip (

tiny-c/8080 Version 80-01-01

294C	CD	1F	22	0380	CALL	LIT	
294F	CD	A9	25	0390	CALL	ASGN	;evaluate condition
2952	C8			0400	RZ		;return on error
2953	11	9B	20	0410	LXI	D,RPAR	;skip)
2956	CD	1F	22	0420	CALL	LIT	
2959	CD	83	21	0430	CALL	TOPTOI	;condition value
295C	7A			0440	MOV	A,D	
295D	B3			0450	ORA	E	
295E	CA	6E	29	0460	JZ	IF2	
2961	CD	FD	28	0470	CALL	ST	;true, execute conditional
2964	11	66	20	0480	LXI	D,XELS	;skip else clause if there
2967	CD	1F	22	0490	CALL	LIT	
296A	C4	2C	28	0500	CNZ	SKIPST	
296D	C9			0510	RET		
296E	CD	2C	28	0520	CALL	SKIPST	;false, skip conditional
2971	11	66	20	0530	LXI	D,XELS	;execute else clause if there
2974	CD	1F	22	0540	CALL	LIT	
2977	C4	FD	28	0550	CNZ	ST	
297A	C9			0560	RET		
297B	11	74	20	0570	LXI	D,XWHI	;test for WHILE
297E	CD	1F	22	0580	CALL	LIT	
2981	CA	CF	29	0590	JZ	TSEM	
2984	11	99	20	0600	LXI	D,LPAR	;skip (
2987	CD	1F	22	0610	CALL	LIT	
298A	CD	A9	25	0620	CALL	ASGN	;condition
298D	C8			0630	RZ		;return on error
298E	11	9B	20	0640	LXI	D,RPAR	;skip)
2991	CD	1F	22	0650	CALL	LIT	
2994	CD	83	21	0660	CALL	TOPTOI	;condition value
2997	7A			0670	MOV	A,D	
2998	B3			0680	ORA	E	
2999	CA	CB	29	0690	JZ	WH2	
299C	2A	5A	20	0700	LHLD	STCURS	;true, save STCURS and CURSOR
299F	E5			0710	PUSH	H	
29A0	2A	5C	20	0720	LHLD	CURSOR	
29A3	E5			0730	PUSH	H	
29A4	CD	FD	28	0740	CALL	ST	;execute object of while
29A7	E1			0750	POP	H	;saved cursor into OBJT

29A8	22	21	2A	0760		SHLD	OBJT	
29AB	E1			0770		POP	H	; and stcurs into AGIN
29AC	22	23	2A	0780		SHLD	AGIN	
29AF	3A	4D	20	0790		LDA	BRAKE	;if a BREAK statement caused
29B2	B7			0800		ORA	A	; this return, then set CURSOR
29B3	CA	C4	29	0810		JZ	WH3	; to object of the while and
29B6	2A	21	2A	0820		LHLD	OBJT	; skip over it, and restore
29B9	22	5C	20	0830		SHLD	CURSOR	; break. The WHILE is alllll
29BC	CD	2C	28	0840		CALL	SKIPST	; done.
29BF	AF			0850		XRA	A	
29C0	32	4D	20	0860		STA	BRAKE	
29C3	C9			0870		RET		
29C4	2A	23	2A	0880	WH3	LHLD	AGIN	;Otherwise, set cursor back to
29C7	22	5C	20	0890		SHLD	CURSOR	; beginning of while statement
29CA	C9			0900		RET		; and return, causing WHILE to
29CB				0910				to be done again.
29CB	CD	2C	28	0920	WH2	CALL	SKIPST	;If condition is false, skip
29CE	C9			0930		RET		; the object, and done.
29CF	11	A4	20	0940	TSEM	LXI	D,SEMI	;test for null statement
29D2	CD	1F	22	0950		CALL	LIT	
29D5	C2	43	23	0960		JNZ	REM	;and done
29D8	11	7A	20	0970	TRET	LXI	D,XRET	;test for RETURN statement
29DB	CD	1F	22	0980		CALL	LIT	
29DE	CA	FB	29	0990		JZ	TBRK	
29E1	00	00		1000		NOP	NOP	
29E3	00	00	00	1010		NOP	NOP	NOP
29E6	11	A4	20	1020		LXI	D,SEMI	;if ; of remark push a 0.
29E9	CD	1F	22	1030		CALL	LIT	
29EC	C2	B9	30	1040		JNZ	XX2	
29EF	11	BE	20	1050		LXI	D,XNL	
29F2	CD	1F	22	1060		CALL	LIT	
29F5	C2	B9	30	1070		JNZ	XX2	
29F8	C3	B3	30	1080		JMP	XX	
29FB	11	81	20	1090	TBRK	LXI	D,XBRK	;test for BREAK
29FE	CD	1F	22	1100		CALL	LIT	
2A01	CA	0A	2A	1110		JZ	TASG	
2A04	3E	01		1120		MVI	A,1	;set break flag
2A06	32	4D	20	1130		STA	BRAKE	

tiny-c/8080 Version 80-01-01

2A09 C9	1140		RET		
2A0A CD A9 25	1150	TASG	CALL	ASGN	;if none of above, must be an
2A0D CA 1C 2A	1160		JZ	STER	; expression, or an error.
2A10 CD 83 21	1170		CALL	TOPTOI	;if an expression, discard its
2A13	1180				value.
2A13 11 A4 20	1190		LXI	D,SEMI	;skip optional ;
2A16 CD 1F 22	1200		CALL	LIT	
2A19 C3 43 23	1210		JMP	REM	;and done
2A1C CD F1 21	1220	STER	CALL	ESET	
2A1F 01	1230		DB	STATERR	;statement error
2A20 C9	1240		RET		
2A21 00 00	1250	OBJT	DW	0	;points to object of while
2A23 00 00	1260	AGIN	DW	0	;points to beginning of while
2A25	1270				
2A25	1280				
2A25 11 6F 20	1290				
2A28 CD 1F 22	1300	DECL	LXI	D,XCHAR	
2A2B CA 49 2A	1310		CALL	LIT	;test for CHAR
2A2E 3E 43	1320	CH2	JZ	TINT	
2A30 21 00 00	1330		MVI	A,'C'	
2A33 CD 8F 28	1340		LXI	H,0	
2A36 11 9D 20	1350		CALL	VALLOC	
2A39 CD 1F 22	1360		LXI	D,COMMA	
2A3C C2 2E 2A	1370		CALL	LIT	
2A3F 11 A4 20	1380		JNZ	CH2	;get all vars
2A42 CD 1F 22	1390	CH3	LXI	D,SEMI	;skip optional ;
2A45 3E 7F	1400		CALL	LIT	
2A47 B7	1410		MVI	A,07FH	;set flag to Not Zero
2A48 C9	1420		ORA	A	
2A49 11 6B 20	1430		RET		
2A4C CD 1F 22	1440	TINT	LXI	D,XINT	
2A4F C8	1450		CALL	LIT	
2A50 3E 49	1460		RZ		;flag is zero
2A52 21 00 00	1470	IN2	MVI	A,'I'	
2A55 CD 8F 28	1480		LXI	H,0	
2A58 11 9D 20	1490		CALL	VALLOC	
2A5B CD 1F 22	1500		LXI	D,COMMA	
2A5E C2 50 2A	1510		CALL	LIT	
			JNZ	IN2	

tiny-c/8080 Version 80-01-01

2A61 C3 3F 2A	1520	JMP	CH3	
2A64	1530			;
2A64	1540			;catches interrupts (ESC key) at appl level.
2A64 3A 62 20	1550	QUIT	LDA	APPLVL
2A67 B7	1560		ORA	A
2A68 C8	1570		RZ	
2A69 CD 10 20	1580		CALL	CHRDY
2A6C C8	1590		RZ	
2A6D 47	1600	MOV	B,A	;char keyed in -> B
2A6E 3A 35 20	1610	LDA	ESCAPE	
2A71 B8	1620	CMP	B	
2A72 C0	1630	RNZ		
2A73 CD 0A 20	1640	CALL	INCH	;discard the ESC
2A76 CD F1 21	1650	CALL	ESET	;signal the escape
2A79 63	1660	DB	KILL	
2A7A C9	1670	RET		
2A7B	8888	END7	EQU	\$

tiny-c/8080 Version 80-01-01

tiny-c/8080 Version 80-01-01

2A7B		0000	;		
2A7B		0010	;evaluates arguments of a function. Sets cursor to		
2A7B		0020	; beginning of function's text. Parses its argument		
2A7B		0030	; declarations, giving them values of the parameters.		
2A7B		0040	; executes the function. Determines cause of exit, and		
2A7B		0050	; pushes default 0 return value if needed. Restores		
2A7B		0060	; cursor.		
2A7B	00	0070	NARGS	DB	0 ;number of args
2A7C	00 00	0080	WHERE	DW	0 ;0 for MC, otherwise address of
2A7E		0090	; function.		
2A7E	00 00	0100	ARG	DW	0 ;pointer into stack to first
2A80		0110	; arg.		
2A80	22 7C 2A	0120	ENTER	SHLD	WHERE
2A83	AF	0130		XRA	A
2A84	32 7B 2A	0140		STA	NARGS
2A87	2A 4E 20	0150		LHLD	TOP
2A8A	11 05 00	0160		LXI	D,5
2A8D	19	0170		DAD	D
2A8E	22 7E 2A	0180		SHLD	ARG
2A91	11 99 20	0190		LXI	D,LPAR ;skip optional (
2A94	CD 1F 22	0200		CALL	LIT
2A97	11 9B 20	0210		LXI	D,RPAR ;test for no args, several ways
2A9A	CD 1F 22	0220		CALL	LIT
2A9D	C2 E9 2A	0230		JNZ	ARGSDNE
2AA0	2A 5C 20	0240		LHLD	CURSOR
2AA3	7E	0250		MOV	A,M
2AA4	FE 5D	0260		CPI	' '
2AA6	CA E9 2A	0270		JZ	ARGSDNE
2AA9	FE 3B	0280		CPI	','
2AAB	CA E9 2A	0290		JZ	ARGSDNE
2AAE	FE 0D	0300		CPI	ASCRET
2AB0	CA E9 2A	0310		JZ	ARGSDNE
2AB3	FE 2F	0320		CPI	'/'
2AB5	CA E9 2A	0330		JZ	ARGSDNE
2AB8	3A 48 20	0340	EN2	LDA	ERR ;eval args, first test for err
2ABB	B7	0350		ORA	A
2ABC	C0	0360		RNZ	
2ABD	2A 7E 2A	0370		LHLD	ARG ;save locals

2AC0 E5	0380		PUSH	H	
2AC1 2A 7C 2A	0390		LHLD	WHERE	
2AC4 E5	0400		PUSH	H	
2AC5 2A 7B 2A	0410		LHLD	NARGS	
2AC8 E5	0420		PUSH	H	
2AC9 CD A9 25	0430		CALL	ASGN	;evaluate
2ACC E1	0440		POP	H	;restore locals
2ACD 7D	0450		MOV	A,L	
2ACE E1	0460		POP	H	
2ACF 22 7C 2A	0470		SHLD	WHERE	
2AD2 E1	0480		POP	H	
2AD3 22 7E 2A	0490		SHLD	ARG	
2AD6 3C	0500		INR	A	;increment NARGS
2AD7 32 7B 2A	0510		STA	NARGS	
2ADA 11 9D 20	0520		LXI	D,COMMA	
2ADD CD 1F 22	0530		CALL	LIT	;comma means more args
2AE0 C2 B8 2A	0540		JNZ	EN2	
2AE3 11 9B 20	0550		LXI	D,RPAR	;optional)
2AE6 CD 1F 22	0560		CALL	LIT	
2AE9 3A 48 20	0570	ARGSDNE	LDA	ERR	
2AEC B7	0580		ORA	A	
2AED C0	0590		RNZ		
2AEE 2A 7C 2A	0600		LHLD	WHERE	;test for MC
2AF1 7C	0610		MOV	A,H	
2AF2 B5	0620		ORA	L	
2AF3 C2 FD 2A	0630		JNZ	EN3	
2AF6 3A 7B 2A	0640		LDA	NARGS	
2AF9 CD BE 2E	0650		CALL	MC	
2AFC C9	0660		RET		
2AFD 2A 5C 20	0670	EN3	LHLD	CURSOR	;save current cursor
2B00 E5	0680		PUSH	H	
2B01 2A 5A 20	0690		LHLD	STCURS	
2B04 E5	0700		PUSH	H	
2B05 2A 7C 2A	0710		LHLD	WHERE	;set cursor to start of fctn
2B08 22 5C 20	0720		SHLD	CURSOR	
2B0B CD B0 23	0730		CALL	NEWFUN	;new layer of value space
2B0E CD 43 23	0740	EN4	CALL	REM	;parse arg decls and pass value
2B11 11 6B 20	0750		LXI	D,XINT	;works just like DECL, except

tiny-c/8080 Version 80-01-01

2B14	CD	1F	22	0760		CALL	LIT	; uses SETARG instead of
2B17	CA	3E	2B	0770		JZ	EN5	; VALLOC.
2B1A	2A	7E	2A	0780	EN6	LHLD	ARG	
2B1D	06	49		0790		MVI	B, 'I'	
2B1F	CD	AF	2B	0800		CALL	SETARG	
2B22	2A	7E	2A	0810		LHLD	ARG	;bump ARG pointer to next
2B25	11	05	00	0820		LXI	D,5	; stack layer
2B28	19			0830		DAD	D	
2B29	22	7E	2A	0840		SHLD	ARG	
2B2C	11	9D	20	0850		LXI	D,COMMA	
2B2F	CD	1F	22	0860		CALL	LIT	
2B32	C2	1A	2B	0870		JNZ	EN6	
2B35	11	A4	20	0880		LXI	D,SEMI	
2B38	CD	1F	22	0890		CALL	LIT	
2B3B	C3	0E	2B	0900		JMP	EN4	
2B3E	11	6F	20	0910	EN5	LXI	D,XCHAR	
2B41	CD	1F	22	0920		CALL	LIT	
2B44	CA	6B	2B	0930		JZ	EN7	
2B47	2A	7E	2A	0940	EN8	LHLD	ARG	
2B4A	06	43		0950		MVI	B, 'C'	
2B4C	CD	AF	2B	0960		CALL	SETARG	
2B4F	2A	7E	2A	0970		LHLD	ARG	
2B52	11	05	00	0980		LXI	D,5	
2B55	19			0990		DAD	D	
2B56	22	7E	2A	1000		SHLD	ARG	
2B59	11	9D	20	1010		LXI	D,COMMA	
2B5C	CD	1F	22	1020		CALL	LIT	
2B5F	C2	47	2B	1030		JNZ	EN8	
2B62	11	A4	20	1040		LXI	D,SEMI	
2B65	CD	1F	22	1050		CALL	LIT	
2B68	C3	0E	2B	1060		JMP	EN4	
2B6B	2A	4E	20	1070	EN7	LHLD	TOP	;test correct number of args
2B6E	11	05	00	1080		LXI	D,5	
2B71	19			1090		DAD	D	
2B72	3A	7E	2A	1100		LDA	ARG	;should be TOP+5
2B75	BD			1110		CMP	L	
2B76	CA	84	2B	1120		JZ	EN9	
2B79	D1			1130		POP	D	;set up old cursor for

2B7A	E1	1140	POP	H	; the error call
2B7B	22 5C 20	1150	SHLD	CURSOR	
2B7E	E5	1160	PUSH	H	
2B7F	D5	1170	PUSH	D	
2B80	CD F1 21	1180	CALL	ESET	
2B83	15	1190	DB	ARGSERR	
2B84	21 7B 2A	1200	LXI	H,NARGS	;pop all args off stack
2B87	35	1210	DCR	M	
2B88	FA 91 2B	1220	JM	EN11	
2B8B	CD A9 21	1230	CALL	POP	
2B8E	C3 84 2B	1240	JMP	EN9	
2B91	3A 48 20	1250	LDA	ERR	;if no errors, execute function
2B94	B7	1260	ORA	A	
2B95	CC FD 28	1270	CZ	ST	
2B98	3A 4C 20	1280	LDA	LEAVE	;push 0 if default leave
2B9B	B7	1290	ORA	A	
2B9C	CC C5 21	1300	CZ	PZERO	
2B9F	AF	1310	XRA	A	;zero LEAVE
2BA0	32 4C 20	1320	STA	LEAVE	
2BA3	E1	1330	POP	H	;restore cvrsor
2BA4	22 5A 20	1340	SHLD	STCURS	
2BA7	E1	1350	POP	H	
2BA8	22 5C 20	1360	SHLD	CURSOR	
2BAB	CD E5 23	1370	CALL	FUNDONE	;pop layer of value space
2BAE	C9	1380	RET		
2BAF		1390			
2BAF		1400			;HL points into stack to an arg. B (used by VALLOC) is
2BAF		1410			; type. SETARG gets actual value of arg, calls VALLOC
2BAF		1420			; to allocate local space, which also puts arg value
2BAF		1430			; into allocated space.
2BAF	C5	1440	SETARG	PUSH B	
2BB0	46	1450	MOV	B,M	;class
2BB1	23	1460	INX	H	
2BB2	7E	1470	MOV	A,M	;lvalue
2BB3	23	1480	INX	H	
2BB4	4E	1490	MOV	C,M	;size
2BB5	23	1500	INX	H	
2BB6	5E	1510	MOV	E,M	;stuff

tiny-c/8080 Version 80-01-01

2BB7	23	1520	INX	H	
2BB8	56	1530	MOV	D,M	
2BB9	FE 41	1540	CPI	'A'	;test for actual
2BBB	CA C2 2B	1550	JZ	SE2	
2BBE	EB	1560	XCHG		;address of datum -> HL
2BBF	5E	1570	MOV	E,M	
2BC0	23	1580	INX	H	
2BC1	56	1590	MOV	D,M	
2BC2	79	1600	MOV	A,C	;if size==1 & class==0
2BC3	3D	1610	DCR	A	
2BC4	B0	1620	ORA	B	
2BC5	C2 CC 2B	1630	JNZ	SE3	
2BC8	7B	1640	MOV	A,E	; then propogate sign
2BC9	07	1650	RLC		
2BCA	9F	1660	SBB	A	
2BCB	57	1670	MOV	D,A	
2BCC	C1	1680	POP	B	;type -> A
2BCD	78	1690	MOV	A,B	
2BCE	EB	1700	XCHG		;passed value -> HL
2BCF	C3 8F 28	1710	JMP	VALLOC	;call/ret, valloc does the rest
2BD2		1720			
2BD2		1730			;scans program and allocates all externals in next fctn
2BD2		1740			; layer. An "endlibrary" line causes a new fctn layer
2BD2		1750			; to be opened.
2BD2	CD B0 23	1760	LINK	CALL	NEWFUN
2BD5	3A 48 20	1770	LI2	LDA	ERR ;check no error
2BD8	B7	1780		ORA	A
2BD9	C0	1790		RNZ	
2BDA	2A 5C 20	1800		LHLD	CURSOR
2BDD	23	1810		INX	H
2BDE	23	1820		INX	H
2BDF	EB	1830		XCHG	
2BE0	2A 60 20	1840		LHLD	PROGEND
2BE3	19	1850		DAD	D
2BE4	EB	1860		XCHG	
2BE5	D8	1870		RC	
2BE6	CD 43 23	1880		CALL	REM ;more text to process, skip
2BE9	11 95 20	1890		LXI	D,LB ; remarks.

2BEC CD 1F 22	1900	CALL	LIT	;test for compound statement.
2BEF CA FC 2B	1910	JZ	LIDCL	
2BF2 06 5B	1920	MVI	B,[''	;skip compound st.
2BF4 0E 5D	1930	MVI	C,']'	
2BF6 CD 54 22	1940	CALL	SKIP	
2BF9 C3 D5 2B	1950	JMP	LI2	
2BFC CD 25 2A	1960	LIDCL CALL	DECL	;test for declaration, and
2BFF C2 D5 2B	1970	JNZ	LI2	; allocate it
2C02 11 87 20	1980	LXI	D,XENDL	;test for endlibrary statement.
2C05 CD 1F 22	1990	CALL	LIT	
2C08 CA 11 2C	2000	JZ	LISYM	
2C0B CD B0 23	2010	CALL	NEWFUN	
2C0E C3 D5 2B	2020	JMP	LI2	
2C11 CD 9A 22	2030	LISYM CALL	SYMNAME	;test for symbol
2C14 CA 46 2C	2040	JZ	LIERR	
2C17 3E 45	2050	MVI	A,'E'	;allocate a variable with
2C19 06 02	2060	MVI	B,2	; class E, size 2, len 1,
2C1B 1E 01	2070	MVI	E,1	; passed value = cursor. (This
2C1D 16 00	2080	MVI	D,0	; is a function entry.)
2C1F 2A 5C 20	2090	LHLD	CURSOR	
2C22 CD 0F 24	2100	CALL	NEWVAR	
2C25 2A 5C 20	2110	LHLD	CURSOR	;advance cursor to beginning of
2C28 3E 5B	2120	MVI	A,[''	; program body.
2C2A BE	2130	LI3 CMP	M	
2C2B CA 3D 2C	2140	JZ	LI4	
2C2E 23	2150	INX	H	
2C2F EB	2160	XCHG		
2C30 2A 60 20	2170	LHLD	PROGEND	
2C33 19	2180	DAD	D	
2C34 EB	2190	XCHG		
2C35 D2 2A 2C	2200	JNC	LI3	
2C38 CD F1 21	2210	CALL	ESET	
2C3B 16	2220	DB	LBRCERR	
2C3C C9	2230	RET		
2C3D 22 5C 20	2240	LI4 SHLD	CURSOR	;skip body
2C40 CD 2C 28	2250	CALL	SKIPST	
2C43 C3 D5 2B	2260	JMP	LI2	
2C46 CD F1 21	2270	LIERR CALL	ESET	

tiny-c/8080 Version 80-01-01

2C49	14	2280	DB	LINKERR
2C4A	C9	2290	RET	
2C4B		2300	;	
2C4B		2310	;move -(bc) bytes from (hl) to (de)	
2C4B	7E	2320	MOVE	MOV A,M
2C4C	12	2330		STAX D
2C4D	13	2340		INX D
2C4E	23	2350		INX H
2C4F	0C	2360		INR C
2C50	C2 4B 2C	2370		JNZ MOVE
2C53	04	2380		INR B
2C54	C2 4B 2C	2390		JNZ MOVE
2C57	C9	2400		RET
2C58		8888	END8	EQU \$

2C58		0000	;it all starts here!!!!
2C58		0010	;cold start erases system level tc programs, and enters
2C58		0020	; the loader. Used to load a tailored or different
2C58		0030	; system program.
2C58		0040	;warm start does not erase sys level progs, but enters
2C58		0050	; the loader so more can be loaded.
2C58		0060	;hot start assumes all the loading is done, and immed
2C58		0070	; starts up the loaded sys level tc prog.
2C58		0080	;Unfortunately, there is no hot start that preserves
2C58		0090	; application programs.
2C58	2A 46 20	0100	COLD LHL D MSTACK ;initialize 8080 stack, if need
2C5B	7C	0110	MOV A,H
2C5C	B5	0120	ORA L
2C5D	CA 61 2C	0130	JZ \$+1
2C60	F9	0140	SPHL
2C61	01 F6 FF	0150	LXI B,-10 ;copy initial statement
2C64	2A 42 20	0160	LHL D BPR ; PR
2C67	EB	0170	XCHG
2C68	21 08 2D	0180	LXI H,INST ; into PR
2C6B	CD 4B 2C	0190	CALL MOVE
2C6E	2A 42 20	0200	LHL D BPR
2C71	11 09 00	0210	LXI D,9
2C74	19	0220	DAD D
2C75	CD C0 2D	0230	CALL HLNEG
2C78	22 60 20	0240	SHLD PROGEND
2C7B	CD 04 2E	0250	CALL LOGO
2C7E	CD 12 2D	0260	WARM CALL LOADER
2C81	CD 04 2E	0270	HOT CALL LOGO
2C84	2A 60 20	0280	LHL D PROGEND
2C87	CD C0 2D	0290	CALL HLNEG
2C8A	22 5E 20	0300	SHLD PRUSED
2C8D	2A 42 20	0310	LHL D BPR
2C90	22 5C 20	0320	SHLD CURSOR
2C93	2A 3A 20	0330	LHL D BFUN
2C96	11 06 00	0340	LXI D,6
2C99	19	0350	DAD D
2C9A	22 54 20	0360	SHLD CURGLBL
2C9D	11 F4 FF	0370	LXI D,-12

tiny-c/8080 Version 80-01-01

2CA0	19	0380
2CA1	22 52 20	0390
2CA4	2A 3E 20	0400
2CA7	22 50 20	0410
2CAA	2A 36 20	0420
2CAD	11 FB FF	0430
2CB0	19	0440
2CB1	22 4E 20	0450
2CB4	AF	0460
2CB5	67	0470
2CB6	6F	0480
2CB7	32 48 20	0490
2CBA	22 4A 20	0500
2CBD	32 4C 20	0510
2CC0	32 4D 20	0520
2CC3	CD D2 2B	0530
2CC6	CD B0 23	0540
2CC9	2A 42 20	0550
2CCC	22 5C 20	0560
2CCF	CD 22 20	0570
2CD2	CD FD 28	0580
2CD5	CD 28 20	0590
2CD8	21 00 2D	0600
2CDB	CD 15 22	0610
2CDE	3A 48 20	0620
2CE1	B7	0630
2CE2	CA F8 2C	0640
2CE5	2A 48 20	0650
2CE8	EB	0660
2CE9	CD C8 2D	0670
2CEC	3E 20	0680
2CEE	CD 0D 20	0690
2CF1	2A 4A 20	0700
2CF4	EB	0710
2CF5	CD C8 2D	0720
2CF8	3E 0D	0730
2CFA	CD 0D 20	0740
2CFD	C3 7E 2C	0750

NOERR

DAD	D
SHLD	CURFUN
LHLD	BVAR
SHLD	NXTVAR
LHLD	BSTACK
LXI	D,-5
DAD	D
SHLD	TOP
XRA	A
MOV	H,A
MOV	L,A
STA	ERR
SHLD	ERRAT
STA	LEAVE
STA	BRAKE
CALL	LINK
CALL	NEWFUN
LHLD	BPR
SHLD	CURSOR
CALL	PRBEGIN
CALL	ST
CALL	PRDONE
LXI	H,DONE
CALL	PS
LDA	ERR
ORA	A
JZ	NOERR
LHLD	ERR
XCHG	
CALL	PN
MVI	A,' ' ; and a space,
CALL	OUTCH
LHLD	ERRAT
XCHG	
CALL	PN
MVI	A,0DH
CALL	OUTCH
JMP	WARM

;this executes the system prog

2D00 0D	0760	DONE	DB	0DH	
2D01 0D	0770		DB	0DH	
2D02 44	0780		DB	'D'	
2D03 4F	0790		DB	'O'	
2D04 4E	0800		DB	'N'	
2D05 45	0810		DB	'E'	
2D06 20	0820		DB	' '	
2D07 00	0830		DB	0	
2D08 5B	0840	INST	DB	'['	
2D09 6D	0850		DB	'm'	
2D0A 61	0860		DB	'a'	
2D0B 69	0870		DB	'i'	
2D0C 6E	0880		DB	'n'	
2D0D 28	0890		DB	'('	
2D0E 29	0900		DB	')'	
2D0F 3B	0910		DB	','	
2D10 5D	0920		DB	']'	
2D11 00	0930		DB	0	
2D12	0940				
2D12 21 98 2D	0950	; LOADER	LXI	H,BUFF	
2D15 3E 3E	0960		MVI	A,'>'	
2D17 CD 0D 20	0970		CALL	OUTCH	
2D1A CD 0D 20	0980		CALL	OUTCH	
2D1D CD 0D 20	0990		CALL	OUTCH	
2D20 CD 0A 20	1000	D2	CALL	INCH	
2D23 C3 D1 30	1010		JMP	XX4	
2D26 78	1020	D4	MOV	A,B	
2D27 CC 0D 20	1030		CZ CNZ	OUTCH	
2D2A 77	1040		MOV	M,A	
2D2B FE 7F	1050		CPI	7FH	;delete char
2D2D CA 39 2D	1060		JZ	D3	
2D30 FE 0D	1070		CPI	0DH	;return
2D32 CA 49 2D	1080		JZ	DOIT	
2D35 23	1090		INX	H	
2D36 C3 20 2D	1100		JMP	D2	
2D39 11 67 D2	1110	D3	LXI	D,-BUFF-1	
2D3C E5	1120		PUSH	H	
2D3D 19	1130		DAD	D	

tiny-c/8080 Version 80-01-01

2D3E	E1		1140		POP	H	
2D3F	D2	20 2D	1150		JNC	D2	
2D42	2B		1160		DCX	H	
2D43	C3	20 2D	1170		JMP	D2	
2D46	C3	20 2D	1180		JMP	D2	
2D49	36	00	1190	DOIT	MVI	M,0	;null at command's end
2D4B	3A	99 2D	1200		LDA	BUFF+1	;ignore period in buff.
2D4E	47		1210		MOV	B,A	
2D4F	3A	92 20	1220		LDA	XR	;the letter r
2D52	B8		1230		CMP	B	
2D53	CA	6E 2D	1240		JZ	LOAD	
2D56	C3	EE 30	1250		JMP	XX8	
2D59	B8		1260	L4	CMP	B	
2D5A	C8		1270		RZ		;leaves editor
2D5B	3E	3F	1280		MVI	A,'?'	;unrecognized command
2D5D	CD	0D 20	1290		CALL	OUTCH	
2D60	CD	0D 20	1300		CALL	OUTCH	
2D63	CD	0D 20	1310		CALL	OUTCH	
2D66	3E	0D	1320		MVI	A,0DH	
2D68	CD	0D 20	1330		CALL	OUTCH	
2D6B	C3	12 2D	1340		JMP	LOADER	
2D6E	21	9B 2D	1350	LOAD	LXI	H,BUFF+3	;file name
2D71	11	01 00	1360		LXI	D,1	;read option
2D74	01	01 00	1370		LXI	B,1	;unit
2D77	C3	E0 30	1380		JMP	XX6	
2D7A	C2	12 2D	1390	L3	JNZ	LOADER	
2D7D	2A	60 20	1400		LHLD	PROGEND	;where to load (stored neg)
2D80	CD	C0 2D	1410	L2	CALL	HLNEG	
2D83	01	01 00	1420		LXI	B,1	;unit
2D86	CD	16 20	1430		CALL	FREAD	;read one block
2D89	C2	E8 30	1440		JNZ	XX7	;err or end of file
2D8C	19		1450		DAD	D	;# bytes read in DE
2D8D	36	00	1460		MVI	M,0	;just beyond last byte read
2D8F	CD	C0 2D	1470		CALL	HLNEG	
2D92	22	60 20	1480		SHLD	PROGEND	;points to null byte at end
2D95	C3	80 2D	1490		JMP	L2	
2D98			1500	BUFF	DS	40	
2DC0			1510				

2DC0		1520	;Negate HL	HL	
2DC0	7C	1530	HLNEG	MOV	A,H
2DC1	2F	1540		CMA	
2DC2	67	1550		MOV	H,A
2DC3	7D	1560		MOV	A,L
2DC4	2F	1570		CMA	
2DC5	6F	1580		MOV	L,A
2DC6	23	1590		INX	H
2DC7	C9	1600		RET	
2DC8		1610			
2DC8		1620	;print (DE) as signed integer		
2DC8	21 98 2D	1630	PN	LXI	H,BUFF
2DCB	CD D6 2D	1640		CALL	ITOA
2DCE	36 00	1650		MVI	M,0 ;put null at end
2DD0	21 98 2D	1660		LXI	H,BUFF
2DD3	C3 15 22	1670		JMP	PS ;and done
2DD6		1680			
2DD6		1690	;convert (DE) to ascii signed integer		
2DD6	7A	1700	ITOA	MOV	A,D ;test for minus
2DD7	B7	1710		ORA	A
2DD8	F2 E1 2D	1720		JP	NTOA
2DDB	CD 74 21	1730		CALL	DENEG ;make positive
2DDE	36 2D	1740		MVI	M,'-' ;output minus
2DE0	23	1750		INX	H ;now fall into NTOA
2DE1		1760	;convert (DE) to ascii unsigned integer		
2DE1	7A	1770	NTOA	MOV	A,D
2DE2	B3	1780		ORA	E ;must be at least one digit, so
2DE3	C2 EA 2D	1790		JNZ	NT2 ; test for 0.
2DE6	36 30	1800		MVI	M,'0'
2DE8	23	1810		INX	H
2DE9	C9	1820		RET	
2DEA	AF	1830	NT2	XRA	A ;put mark on stack
2DEB	F5	1840		PUSH	P
2DEC	01 0A 00	1850	NT3	LXI	B,10
2DEF	E5	1860		PUSH	H
2DF0	CD 6F 21	1870		CALL	DDIV
2DF3	7D	1880		MOV	A,L ;remainder -> A
2DF4	E1	1890		POP	H

tiny-c/8080 Version 80-01-01

2DF5 C6 30	1900	ADI	'0'	
2DF7 F5	1910	PUSH	P	;ascii digit -> stack
2DF8 7A	1920	MOV	A,D	;done if quotient is zero
2DF9 B3	1930	ORA	E	
2DFA C2 EC 2D	1940	JNZ	NT3	
2DFD F1	1950	POP	P	;top of stack is digit or mark.
2DFE C8	1960	RZ		;done if mark.
2DFF 77	1970	MOV	M,A	;otherwise digit -> buffer.
2E00 23	1980	INX	H	
2E01 C3 FD 2D	1990	JMP	NT4	
2E04	2000			
2E04	2010	;prints the	copyright message on the terminal.	
2E04 21 0A 2E	2020	LOGO	LXI	H,CPMSG
2E07 C3 15 22	2030		JMP	PS
2E0A 0C	2040	CPMSG	DB	0CH
2E0B 2A	2050		DB	'*'
2E0C 2A	2060		DB	'*'
2E0D 2A	2070		DB	'*'
2E0E 20	2080		DB	' '
2E0F 20	2090		DB	' '
2E10 54	2100		DB	'T'
2E11 49	2110		DB	'I'
2E12 4E	2120		DB	'N'
2E13 59	2130		DB	'Y'
2E14 2D	2140		DB	'-'
2E15 43	2150		DB	'C'
2E16 20	2160		DB	' '
2E17 20	2170		DB	' '
2E18 20	2180		DB	' '
2E19 56	2190		DB	'V'
2E1A 45	2200		DB	'E'
2E1B 52	2210		DB	'R'
2E1C 53	2220		DB	'S'
2E1D 49	2230		DB	'I'
2E1E 4F	2240		DB	'O'
2E1F 4E	2250		DB	'N'
2E20 20	2260		DB	' '
2E21 31	2270		DB	'l'

2E22 2E	2280	DB	'.'
2E23 30	2290	DB	'0'
2E24 20	2300	DB	' '
2E25 20	2310	DB	' '
2E26 2A	2320	DB	'*'
2E27 2A	2330	DB	'*'
2E28 2A	2340	DB	'*'
2E29 0D	2350	DB	0DH
2E2A 0A	2360	DB	0AH
2E2B 43	2370	DB	'C'
2E2C 4F	2380	DB	'O'
2E2D 50	2390	DB	'P'
2E2E 59	2400	DB	'Y'
2E2F 52	2410	DB	'R'
2E30 49	2420	DB	'I'
2E31 47	2430	DB	'G'
2E32 48	2440	DB	'H'
2E33 54	2450	DB	'T'
2E34 20	2460	DB	' '
2E35 31	2470	DB	'1'
2E36 39	2480	DB	'9'
2E37 37	2490	DB	'7'
2E38 37	2500	DB	'7'
2E39 2C	2510	DB	'.'
2E3A 20	2520	DB	' '
2E3B 54	2530	DB	'T'
2E3C 20	2540	DB	' '
2E3D 41	2550	DB	'A'
2E3E 20	2560	DB	' '
2E3F 47	2570	DB	'G'
2E40 49	2580	DB	'I'
2E41 42	2590	DB	'B'
2E42 53	2600	DB	'S'
2E43 4F	2610	DB	'O'
2E44 4E	2620	DB	'N'
2E45 0D	2630	DB	0DH
2E46 0A	2640	DB	0AH
2E47 00	2650	DB	0
2E48	8888	END9 EQU	\$

tiny-c/8080 Version 80-01-01

2E48		0000		;move the block (DE)...(HL) inclusive (BC) bytes. If
2E48		0010		; (BC) is positive, the block is moved up in RAM,
2E48		0020		; highest byte first, lowest byte last. If (BC) is
2E48		0030		; negative, the block is moved down in RAM, lowest
2E48		0040		; byte first. Thus large blocks can be safely moved
2E48		0050		; up or down short distances.
2E48	78	0060	MOVEBL	MOV A,B
2E49	B7	0070		ORA A
2E4A	FA 70 2E	0080		JM MOVEDN
2E4D	B1	0090		ORA C
2E4E	C8	0100		RZ
2E4F	22 87 2E	0110	MOVEUP	SHLD FROMPTR ;hi end of block is fromptr
2E52	09	0120		DAD B ;to pointer -> DE
2E53	EB	0130		XCHG
2E54	3A 87 2E	0140		LDA FROMPTR ; - length -> BC
2E57	2F	0150		CMA
2E58	85	0160		ADD L ; - length =
2E59	4F	0170		MOV C,A ; current HL - fromptr +1
2E5A	3A 88 2E	0180		LDA FROMPTR+1
2E5D	2F	0190		CMA
2E5E	8C	0200		ADC H
2E5F	47	0210		MOV B,A
2E60	2A 87 2E	0220		LHLD FROMPTR
2E63	7E	0230	MU2	MOV A,M
2E64	12	0240		STAX D
2E65	2B	0250		DCX H
2E66	1B	0260		DCX D
2E67	0C	0270		INR C
2E68	C2 63 2E	0280		JNZ MU2
2E6B	04	0290		INR B
2E6C	C2 63 2E	0300		JNZ MU2
2E6F	C9	0310		RET
2E70	EB	0320	MOVEDN	XCHG ;lo end of block is from ptr
2E71	22 87 2E	0330		SHLD FROMPTR
2E74	09	0340		DAD B ;to pointer -> HL
2E75	3A 87 2E	0350		LDA FROMPTR ; - length -> BC
2E78	93	0360		SUB E
2E79	4F	0370		MOV C,A

2E7A	3A	88	2E	0380	LDA	FROMPTR+1
2E7D	9A			0390	SBB	D
2E7E	47			0400	MOV	B,A
2E7F	0B			0410	DCX	B
2E80	EB			0420	XCHG	;to ptr -> DE
2E81	2A	87	2E	0430	LHLD	FROMPTR ;from ptr -> HL
2E84	C3	4B	2C	0440	JMP	MOVE
2E87	00	00		0450	FROMPTR DW	0
2E89				0460		
2E89				0470		;scan for the Nth occurrence of a character in a block,
2E89				0480		; or the end of the block, whichever comes first. The
2E89				0490		; block is (DE)..(HL) inclusive. N is (BC) and can be
2E89				0500		; 0 to 65k. (A) is the character. On completion, (DE)
2E89				0510		; points to the Nth occurrence, or to the last byte of
2E89				0520		; the block. (BC) is N minus the number of (A) found,
2E89				0530		; e.g. 0 if N (A)'s were found. HL is undisturbed.
2E89	F5			0540	SCANN PUSH	P ;ch -> stack
2E8A	EB			0550	XCHG	;reverse first and last
2E8B	79			0560	SC2 MOV	A,C
2E8C	B0			0570	ORA	B ;test if done
2E8D	CA	A2	2E	0580	JZ	SC9
2E90	7B			0590	MOV	A,E
2E91	95			0600	SUB	L
2E92	7A			0610	MOV	A,D
2E93	9C			0620	SBB	H
2E94	DA	A2	2E	0630	JC	SC9
2E97	F1			0640	POP	P
2E98	F5			0650	PUSH	P
2E99	BE			0660	CMP	M
2E9A	C2	9E	2E	0670	JNZ	SC9
2E9D	0B			0680	DCX	B
2E9E	23			0690	INX	H
2E9F	C3	8B	2E	0700	JMP	SC2
2EA2	2B			0710	SC9 DCX	H
2EA3	EB			0720	XCHG	
2EA4	F1			0730	POP	P
2EA5	C9			0740	RET	
2EA6				0750		

tiny-c/8080 Version 80-01-01

```
2EA6      0760
2EA6      0770
2EA6      0780
2EA6      0790
2EA6 01 00 00 0800
2EA9 F5      0810
2EAA 7D      0820
2EAB 93      0830
2EAC 7C      0840
2EAD 9A      0850
2EAE DA BC 2E 0860
2EB1 F1      0870
2EB2 F5      0880
2EB3 BE      0890
2EB4 2B      0900
2EB5 C2 AA 2E 0910
2EB8 03      0920
2EB9 C3 AA 2E 0930
2EBC F1      0940
2EBD C9      0950
2EBE      8888
```

;count the occurrences of a character in a block. (A) is
; the character. The block is (DE)..(HL) inclusive.
; The count is returned in (BC). (A) and (DE) are
; unchanged. (HL) is clobbered.

```
COUNTCH LXI    B,0
          PUSH  P      ;ch -> stack
CC2      MOV   A,L      ;test for end
          SUB   E
          MOV   A,H
          SBB   D
          JC    CC9
          POP   P
          PUSH  P
          CMP   M
          DCX   H
          JNZ   CC2
          INX   B      ;count this one
          JMP   CC2
CC9      POP   P
          RET
END10    EQU    $
```


2EBE		0000	;Machine Call routine to interface to 8080 coded
2EBE		0010	; routines. Standard routines used by the system
2EBE		0020	; are coded here, numbers 1 to 11. 12 to 999 are
2EBE		0030	; reserved. 1000 and up are available to users.
2EBE	32 34 20	0040	MC STA MCARGS ;for checking,
2EC1	CD 83 21	0050	CALL TOPTOI ; for MC's that need it.
2EC4	21 18 FC	0060	LXI H,-1000 ;test for user MC
2EC7	19	0070	DAD D
2EC8	DA 1F 20	0080	JC USERMC
2ECB	7B	0090	MOV A,E ;fctn num -> A
2ECC	FE 01	0100	CPI 1
2ECE	CA 17 2F	0110	JZ MC1
2ED1	FE 02	0120	CPI 2
2ED3	CA 21 2F	0130	JZ MC2
2ED6	FE 03	0140	CPI 3
2ED8	CA 44 2F	0150	JZ MC3
2EDB	FE 04	0160	CPI 4
2EDD	CA 62 2F	0170	JZ MC4
2EE0	FE 05	0180	CPI 5
2EE2	CA 78 2F	0190	JZ MC5
2EE5	FE 06	0200	CPI 6
2EE7	CA 90 2F	0210	JZ MC6
2EEA	FE 07	0220	CPI 7
2EEC	CA 9B 2F	0230	JZ MC7
2EEF	FE 08	0240	CPI 8
2EF1	CA AE 2F	0250	JZ MC8
2EF4	FE 09	0260	CPI 9
2EF6	CA C4 2F	0270	JZ MC9
2EF9	FE 0A	0280	CPI 10
2EFB	CA EE 2F	0290	JZ MC10
2EFE	FE 0B	0300	CPI 11
2F00	CA F0 2F	0310	JZ MC11
2F03	FE 0C	0320	CPI 12
2F05	CA 81 30	0330	JZ MC12
2F08	FE 0D	0340	CPI 13
2F0A	CA 8A 30	0350	JZ MC13
2F0D	FE 0E	0360	CPI 14
2F0F	CA A8 30	0370	JZ MC14

tiny-c/8080 Version 80-01-01

2F12	CD	F1	21	0380	MCESET	CALL	ESET	
2F15	18			0390		DB	MCERR	
2F16	C9			0400		RET		
2F17				0410				
2F17				0420				
2F17	CD	83	21	0430				
2F1A	CD	C8	21	0440	MC1	CALL	TOPTOI	;char -> A
2F1D	7B			0450		CALL	PUSHK	;push it back
2F1E	C3	0D	20	0460		MOV	A,E	
2F21				0470		JMP	OUTCH	
2F21				0480				
2F21	CD	0A	20	0490				
2F24	47			0500	MC2	CALL	INCH	;char -> DE
2F25	3A	62	20	0510		MOV	B,A	;test for ESC in appl level
2F28	B7			0520		LDA	APPLVL	
2F29	CA	37	2F	0530		ORA	A	
2F2C	3A	35	20	0540		JZ	USEIT	
2F2F	B8			0550		LDA	ESCAPE	
2F30	C2	37	2F	0560		CMP	B	
2F33	CD	F1	21	0570		JNZ	USEIT	
2F36	63			0580		CALL	ESET	
2F37	3A	09	20	0590		DB	KILL	
2F3A	B7			0600	USEIT	LDA	ECHO	;test if echo required
2F3B	C3	D9	30	0610		ORA	A	
2F3E	5F			0620		JMP	XX5	
2F3F	AF			0630	U2	MOV	E,A	
2F40	57			0640		XRA	A	
2F41	C3	C8	21	0650		MOV	D,A	
2F44				0660		JMP	PUSHK	;put char onto stack
2F44				0670				
2F44	CD	83	21	0680				
2F47	D5			0690	MC3	CALL	TOPTOI	
2F48	CD	83	21	0700		PUSH	D	
2F4B	D5			0710		CALL	TOPTOI	
2F4C	CD	83	21	0720		PUSH	D	
2F4F	D5			0730		CALL	TOPTOI	
2F50	CD	83	21	0740		PUSH	D	
2F53	7B			0750		CALL	TOPTOI	;r/w -> A
						MOV	A,E	

2F54 B2	0760	ORA	D	
2F55 E1	0770	POP	H	;name pointer -> HL
2F56 D1	0780	POP	D	;file size -> DE
2F57 C1	0790	POP	B	;unit -> BC
2F58 CD 13 20	0800	CALL	FOPEN	
2F5B 11 00 00	0810	LXI	D,0	
2F5E 5F	0820	MOV	E,A	;push result code
2F5F C3 C8 21	0830	JMP	PUSHK	
2F62	0840			
2F62	0850			; read block(where, unit)
2F62 CD 83 21	0860	MC4 CALL	TOPTOI	
2F65 D5	0870	PUSH	D	
2F66 CD 83 21	0880	CALL	TOPTOI	
2F69 EB	0890	XCHG		;where -> HL
2F6A C1	0900	POP	B	;unit -> BC
2F6B CD 16 20	0910	CALL	FREAD	
2F6E CA 75 2F	0920	JZ	MC4P	;if result code is 0 DE has
2F71 11 FF FF	0930	LXI	D,-1	; byte count to be pushed.
2F74 5F	0940	MOV	E,A	; Otherwise A is an err or eof
2F75 C3 C8 21	0950	MC4P JMP	PUSHK	; code to be returned negative
2F78	0960			
2F78	0970			;write block (first byte, last byte, unit). Block may
2F78	0980			; be any size from 1 to 256.
2F78 CD 83 21	0990	MC5 CALL	TOPTOI	
2F7B D5	1000	PUSH	D	
2F7C CD 83 21	1010	CALL	TOPTOI	
2F7F D5	1020	PUSH	D	
2F80 CD 83 21	1030	CALL	TOPTOI	
2F83 EB	1040	XCHG		;first -> HL
2F84 D1	1050	POP	D	;last -> DE
2F85 C1	1060	POP	B	;unit -> BC
2F86 CD 19 20	1070	CALL	FWRITE	
2F89 11 00 00	1080	LXI	D,0	;push result code
2F8C 5F	1090	MOV	E,A	
2F8D C3 C8 21	1100	JMP	PUSHK	
2F90	1110			
2F90	1120			;close file (unit)
2F90 CD 83 21	1130	MC6 CALL	TOPTOI	

tiny-c/8080 Version 80-01-01

2F93 4B	1140	MOV	C,E	;unit -> BC
2F94 42	1150	MOV	B,D	
2F95 CD 1C 20	1160	CALL	FCLOSE	
2F98 C3 C5 21	1170	JMP	PZERO	;return a 0
2F9B	1180			
2F9B	1190			;move a block up or down. Args are first,last,K. If K
2F9B	1200			; negative, block is moved down k bytes, if positive
2F9B	1210			; then up K bytes.
2F9B CD 83 21	1220	MC7	CALL	TOPTOI
2F9E D5	1230		PUSH	D
2F9F CD 83 21	1240		CALL	TOPTOI
2FA2 D5	1250		PUSH	D
2FA3 CD 83 21	1260		CALL	TOPTOI ;first -> DE
2FA6 E1	1270		POP	H ;last
2FA7 C1	1280		POP	B ;K
2FA8 CD 48 2E	1290		CALL	MOVEBL
2FAB C3 C5 21	1300		JMP	PZERO ;return a 0
2FAE	1310			
2FAE	1320			;count # instances of character CH in a block. Args are
2FAE	1330			; first,last,CH.
2FAE CD 83 21	1340	MC8	CALL	TOPTOI
2FB1 D5	1350		PUSH	D
2FB2 CD 83 21	1360		CALL	TOPTOI
2FB5 D5	1370		PUSH	D
2FB6 CD 83 21	1380		CALL	TOPTOI ;first -> DE
2FB9 E1	1390		POP	H ;last
2FBA C1	1400		POP	B ;ch -> A
2FBB 79	1410		MOV	A,C
2FBC CD A6 2E	1420		CALL	COUNTCH
2FBF 59	1430		MOV	E,C ;count -> DE
2FC0 50	1440		MOV	D,B
2FC1 C3 C8 21	1450		JMP	PUSHK
2FC4	1460			
2FC4	1470			;scan for nth occurrence of CH in a block. Args are
2FC4	1480			; first,last,CH,cnt address. Return pointer to nth
2FC4	1490			; occurrence,if it exists, otherwise to last. Also
2FC4	1500			; cnt is reduced by one for every CH found.
2FC4 CD 83 21	1510	MC9	CALL	TOPTOI

2FC7 D5	1520	PUSH	D	
2FC8 CD 83 21	1530	CALL	TOPTOI	
2FCB D5	1540	PUSH	D	
2FCC CD 83 21	1550	CALL	TOPTOI	
2FCF D5	1560	PUSH	D	
2FD0 CD 83 21	1570	CALL	TOPTOI	;first -> DE
2FD3 E1	1580	POP	H	;last
2FD4 C1	1590	POP	B	;ch -> A
2FD5 79	1600	MOV	A,C	
2FD6 E3	1610	XTHL		
2FD7 4E	1620	MOV	C,M	;cnt -> BC
2FD8 23	1630	INX	H	
2FD9 46	1640	MOV	B,M	
2FDA 2B	1650	DCX	H	
2FDB E3	1660	XTHL		;addr of cnt still on stack
2FDC D5	1670	PUSH	D	;first on stack, too
2FDD CD 89 2E	1680	CALL	SCANN	
2FE0 E1	1690	POP	H	;make ptr (DE) relative to
2FE1 7B	1700	MOV	A,E	; first
2FE2 95	1710	SUB	L	
2FE3 5F	1720	MOV	E,A	
2FE4 7A	1730	MOV	A,D	
2FE5 9C	1740	SBB	H	
2FE6 57	1750	MOV	D,A	
2FE7 E1	1760	POP	H	;BC -> cnt
2FE8 71	1770	MOV	M,C	
2FE9 23	1780	INX	H	
2FEA 70	1790	MOV	M,B	
2FEB C3 C8 21	1800	JMP	PUSHK	;return pointer to last byte
2FEE	1810			; examined.
2FEE	1820			
2FEE	1830			;trap to moniter 4.0 for debugging.
2FEE FF	1840	MC10 DB	0FFH	;RST 7
2FEF C9	1850	RET		
2FF0	1860			
2FF0	1870			;enters an application program, setting up a new
2FF0	1880			; globals variable level, redefining progend, links
2FF0	1890			; the program, executes if no error occured, upon

POLY !

tiny-c/8080 Version 80-01-01

2FF0		1900	; completion captures a few facts (err, and either
2FF0		1910	; cursor or errat) and restores old globals level,
2FF0		1920	; progend, zeros err, pushes a zero as the value of
2FF0		1930	; this function, and resumes the calling program.
2FF0 2A 5C 20		1940 MC11 LHL D	CURSOR
2FF3 E5		1950	PUSH H
2FF4 2A 60 20		1960	LHL D PROGEND
2FF7 E5		1970	PUSH H
2FF8 2A 5E 20		1980	LHL D PRUSED
2FFB E5		1990	PUSH H
2FFC 2A 54 20		2000	LHL D CURGLBL
2FFF E5		2010	PUSH H
3000 CD 83 21		2020	CALL TOPTOI ;appl pr address
3003 EB		2030	XCHG
3004 E5		2040	PUSH H
3005 22 5C 20		2050	SHLD CURSOR
3008 CD 83 21		2060	CALL TOPTOI ;end of appl addr
300B EB		2070	XCHG
300C 22 5E 20		2080	SHLD PRUSED
300F CD C0 2D		2090	CALL HLNEG
3012 22 60 20		2100	SHLD PROGEND
3015 CD D2 2B		2110	CALL LINK
3018 2A 52 20		2120	LHL D CURFUN
301B 22 54 20		2130	SHLD CURGLBL
301E CD 83 21		2140	CALL TOPTOI ;start statement address
3021 EB		2150	XCHG
3022 22 5C 20		2160	SHLD CURSOR
3025 CD B0 23		2170	CALL NEWFUN
3028 CD 83 21		2180	CALL TOPTOI ;facts address
302B D5		2190	PUSH D
302C 21 62 20		2200	LXI H,APPLVL ;increment appl level
302F 34		2210	INR M
3030 E5		2220	PUSH H
3031 3A 48 20		2230	LDA ERR ;if no err so far, do it!!
3034 B7		2240	ORA A
3035 C2 41 30		2250	JNZ DONE
3038 CD 22 20		2260	CALL PRBEGIN
303B CD FD 28		2270	CALL ST

303E CD 28 20	2280		CALL	PRDONE	
3041 E1	2290	DONE	POP	H	;its done, decrement appl level
3042 35	2300		DCR	M	
3043 CD E5 23	2310		CALL	FUNDONE	;discard appl locals
3046 CD E5 23	2320		CALL	FUNDONE	; and globals
3049 2A 5C 20	2330		LHLD	CURSOR	;set up facts
304C 3A 48 20	2340		LDA	ERR	
304F B7	2350		ORA	A	
3050 CA 56 30	2360		JZ	\$+3	
3053 2A 4A 20	2370		LHLD	ERRAT	
3056 EB	2380		XCHG		;returned cursor -> DE
3057 E1	2390		POP	H	;facts -> HL
3058 C1	2400		POP	B	;appl pr address -> BC
3059 7B	2410		MOV	A,E	;make returned cursor relative
305A 91	2420		SUB	C	; to appl address
305B 5F	2430		MOV	E,A	
305C 7A	2440		MOV	A,D	
305D 98	2450		SBB	B	
305E 57	2460		MOV	D,A	
305F 3A 48 20	2470		LDA	ERR	
3062 77	2480		MOV	M,A	;err -> facts
3063 AF	2490		XRA	A	
3064 23	2500		INX	H	
3065 77	2510		MOV	M,A	;err hi byte -> facts
3066 23	2520		INX	H	
3067 73	2530		MOV	M,E	;cursor -> facts
3068 23	2540		INX	H	
3069 72	2550		MOV	M,D	
306A E1	2560		POP	H	;curglobal
306B 22 54 20	2570		SHLD	CURGLBL	
306E E1	2580		POP	H	
306F 22 5E 20	2590		SHLD	PRUSED	
3072 E1	2600		POP	H	;progend
3073 22 60 20	2610		SHLD	PROGEND	
3076 E1	2620		POP	H	;cursor
3077 22 5C 20	2630		SHLD	CURSOR	
307A AF	2640		XRA	A	;zero the error
307B 32 48 20	2650		STA	ERR	

tiny-c/8080 Version 80-01-01

307E C3 C5 21	2660	JMP	PZERO	;value of MC11
3081	2670			
3081	2680			;test if keyboard char ready, return copy if so,else 0.
3081 CD 10 20	2690	MC12	CALL	CHRDY
3084 16 00	2700		MVI	D,0
3086 5F	2710		MOV	E,A
3087 C3 C8 21	2720		JMP	PUSHK
308A	2730			
308A	2740			;print RAM, from and to addresses are given
308A	2750			; nulls are mapped to quotes
308A CD 83 21	2760	MC13	CALL	TOPTOI
308D D5	2770		PUSH	D
308E CD 83 21	2780		CALL	TOPTOI
3091 EB	2790		XCHG	;from -> HL
3092 D1	2800		POP	D ;to -> DE
3093 7B	2810	LOOP13	MOV	A,E ;test if done
3094 95	2820		SUB	L
3095 7A	2830		MOV	A,D
3096 9C	2840		SBB	H
3097 DA C5 21	2850		JC	PZERO ;done
309A 7E	2860		MOV	A,M
309B B7	2870		ORA	A
309C C2 A1 30	2880		JNZ	\$+2
309F 3E 22	2890		MVI	A,'''
30A1 CD 0D 20	2900		CALL	OUTCH
30A4 23	2910		INX	H
30A5 C3 93 30	2920		JMP	LOOP13
30A8	2930			
30A8	2940			;print a signed integer
30A8 CD 83 21	2950	MC14	CALL	TOPTOI
30AB D5	2960		PUSH	D
30AC CD C8 2D	2970		CALL	PN
30AF D1	2980		POP	D
30B0 C3 C8 21	2990		JMP	PUSHK

30B3 CD A9 25	3000	; patch at 29EC - 29F8
30B6 C3 BC 30	3010	XX CALL ASGN
30B9 CD C5 21	3020	JMP \$+3
30BC 3E 01	3030	XX2 CALL PZERO
30BE 32 4C 20	3040	MVI A,1
30C1 C9	3050	STA LEAVE
		RET
		; patch to SKIPST
30C2 CD 1F 22	3060	XX3 CALL LIT
30C5 C2 4B 28	3070	JNZ SS6
30C8 11 74 20	3080	LXI D,XWHI
30CB CD 1F 22	3090	CALL LIT
30CE C3 48 28	3100	JMP SS7
		; patch to LOADER
30D1 47	3110	XX4 MOV B,A
30D2 3A 09 20	3120	LDA ECHO
30D5 B7	3130	ORA A
30D6 C3 26 2D	3140	JMP D4
		; patch to MC2
30D9 78	3150	XX5 MOV A,B
30DA C4 0D 20	3160	CNZ OUTCH
30DD C3 3E 2F	3170	JMP U2
		; patch to LOADER
30E0 3E 01	3180	XX6 MVI A,1
30E2 CD 13 20	3190	CALL FOPEN
30E5 C3 7A 2D	3200	JMP L3
30E8 CD 1C 20	3210	XX7 CALL FCLOSE
30EB C3 12 2D	3220	JMP LOADER
30EE 3E 78	3230	XX8 MVI A,'X'
30F0 B8	3240	CMP B
30F1 CA 00 00	3250	JZ 0
30F3 3A 93 20	3260	LDA XG
30F6 C3 59 2C	3270	JMP L4

tiny-c/8080 Version 80-01-01

STATERR	0001
CURSERR	0002
SYMERR	0003
RPARERR	0005
RANGERR	0006
CLASERR	0007
SYNXERR	0009
LVALERR	000E
PUSHERR	0010
TMFUERR	0011
TMVRERR	0012
TMVLERR	0013
LINKERR	0014
ARGSERR	0015
LBRCERR	0016
MCERR	0018
SYMERRA	001A
KILL	0063
VLEN	0008
ASCRT	000D
ECHO	2009
INCH	200A
OUTCH	200D
CHRDY	2010
FOPEN	2013
FREAD	2016
FWRITE	2019
FCLOSE	201C
USERMC	201F
PRBEGIN	2022
STBEGIN	2025
PRDONE	2028
XMCESET	202B
XTOPTOI	202E
XPUSHK	2031
MCARGS	2034
ESCAPE	2035
BSTACK	2036

ESTACK	2038
BFUN	203A
EFUN	203C
BVAR	203E
EVAR	2040
BPR	2042
EPR	2044
MSTACK	2046
ERR	2048
ERRAT	204A
LEAVE	204C
BRAKE	204D
TOP	204E
NXTVAR	2050
CURFUN	2052
CURGLBL	2054
FNAME	2056
LNAME	2058
STCURS	205A
CURSOR	205C
PRUSED	205E
PROGEND	2060
APPLVL	2062
BALPHS	2063
XIF	2063
XELS	2066
XINT	206B
XCHAR	206F
XWHI	2074
XRET	207A
XBRK	2081
XENDL	2087
XR	2092
XG	2093
LB	2095
RB	2097
LPAR	2099
RPAR	209B

tiny-c/8080 Version 80-01-01

COMMA	209D
NEWLINE	209F
CMNT	20A1
XSTAR	20A2
SEMI	20A4
XPCNT	20A6
XSLASH	20A8
XPLUS	20AA
XMINUS	20AC
LT	20AE
GT	20B0
NOTEQ	20B2
EQEQ	20B5
XEQ	20B6
GE	20B8
LE	20BB
XNL	20BE
END1	20C0
EQ	20C0
DNEG	20E7
TOPDIF	20EF
DSUB	20F2
DADD	20FC
DMPY	2106
BCRS	211F
DELS	2128
RDEL	2129
DREM	2131
DDIV	216F
DENEG	2174
DCMP	217C
HLLS	2181
TOPTOI	2183
POPTWO	21A0
POP	21A9
PONE	21BF
PZERO	21C5
PUSHK	21C8

tiny-c/8080 Version 80-01-01

PUSH	21CD
ESET	21F1
ZERO	2209
BZAP	220B
PS	2215
END2	221F
LIT	221F
MATCH	223D
BLANKS	2243
SKIP	2254
ALNUM	227D
ALPHA	2285
SYMNAME	229A
CONST	22B9
REM	2343
ATON	235E
ATOI	237E
END3	23B0
NEWFUN	23B0
FUNDONE	23E5
NEWVAR	240F
ADDRVAL	24E1
CANON	256B
END4	25A9
ASGN	25A9
RELN	25CA
EXPR	2652
TERM	26AA
FACTOR	26FB
END5	282C
SKIPST	282C
VALLOC	288F
END6	28FD
ST	28FD
DECL	2A25
QUIT	2A64
END7	2A7B
ENTER	2A80

tiny-c/8080 Version 80-01-01

SETARG	2BAF
LINK	2BD2
MOVE	2C4B
END8	2C58
COLD	2C58
WARM	2C7E
HOT	2C81
LOADER	2D12
HLNEG	2DC0
PN	2DC8
ITOA	2DD6
NTOA	2DE1
END9	2E48
MOVEBL	2E48
MOVEUP	2E4F
MOVEDN	2E70
SCANN	2E89
COUNTCH	2EA6
END10	2EBE
MC	2EBE
MCESET	2F12

WCHSEL	3573
WC	357E
ENDIS	357E
COORLCH	3579
ECWIN	3583
WALERS	3510
WALERS	354E
WALERS	354E
ENDS	3540
ALOV	3DE7
ALOV	3DD8
BB	3DC8
HGWS	3BC8
POWDER	3D73
HOL	3C87
MYEN	3C1E
COGO	3C28
ENDS	3C28
WALS	3C48
PINK	3B03
SELVBC	3B7A

B

tiny-c/11 Version 11-01-01
TCMAIN

```
1          .SBTTL  GLOBALS
2          .NLIST
3          .TITLE  tiny-c/11
4
5          ;
6          ; tiny-c/11 assembly constants
7          ;
8          000036      MAXSTACK=30.
9          000144      MAXVAR=100.
10         000036      MAXFUN=30.
11         047040      MAXPR=20000.
12         000062      MAXDEPTH=50.
13         000010      VLEN=8.
14
15         ;
16         ; tiny-c/11 error codes
17         ;
18         000001      STATERR=1.
19         000002      CURSERR=2.
20         000003      SYMBOLERR=3.
21         000005      RPARERR=5.
22         000006      RANGERR=6.
23         000007      CLASSERR=7.
24         000011      SYNTAXERR=9.
25         000016      LVALERR=14.
26         000017      TOPERR=15.
27         000020      PUSHERR=16.
28         000021      TMFUNERR=17.
29         000022      TMVARERR=18.
30         000023      TMVALERR=19.
31         000024      LINKERR=20.
32         000025      ARGSErr=21.
33         000026      LBRCERR=22.
34         000030      MCERR=24.
35         000031      LOADERR=25.
36         000032      SYMERR=26.
37
38         ;
39         ; tiny-c/11 global literals
40         ;
41         .GLOBL  LBRACE,RBRACE,LPAREN,RPAREN,COMMA,SEMI,LF,MCLIT,ENDLIB
42         .GLOBL  INT,CHAR,IF,ELSE,WHILE,BREAK,RETERN
43         .GLOBL  EQUAL,PLUS,MINUS,TIMES,DIVIDE,MODULO
44         .GLOBL  EQLEQL,NOTEQL,GRTEQL,LESEQL,GRTHN,LESTHN
45
46         ;
47         ; tiny-c/11 global variables
48         ;
49         .GLOBL  ERR,ERRAT,LEAVE,BRAKE
50         .GLOBL  TOP,NXTVAR,CURFUN,CURGLO,FNAME,LNAME
51         .GLOBL  APPLVL,STCURS,CURSOR,PRUSED,PROGEN
```


tiny-c/11 Version 11-01-01
TCMAIN

```
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66  
67  
68  
69  
70  
71  
72  
73  
74  
75  
76
```

```
; tiny-c/11 subroutines  
; .GLOBL TINYC,ADDRVA,ALPHA,ALPHAN,ASGN,ATOI,BLANKS  
; .GLOBL CANON,CEQ,CEQN,CONST,ENTER,ESET  
; .GLOBL EQ,EXPR,FACTOR,FUNDON,LINK,LIT  
; .GLOBL MC,MOVN,NEWFUN,NEWVAR,NUM,POP,PUSH,RELN,REM  
; .GLOBL RETURN,SAVE,SETARG,SKIP,SKIPST,ST,SYMNAM  
; .GLOBL TCTOI,TERM,TOPTOI,VALLOC  
;  
; tiny-c/11 installation vector entries  
; .GLOBL INCH,OUTCH,FLOAD,FOPEN,FREAD,FWRITE,FCLOSE  
; .GLOBL USERMC,PRBEGN,STBEGN,PRDONE  
; .GLOBL BSTACK,ESTACK,BVAR,EVAR,BFUN,EFUN,BPR,EPR,MSTACK  
;  
; tiny-c/11 register assignments  
;  
R0=%^O0  
R1=%^O1  
R2=%^O2  
R3=%^O3  
R4=%^O4  
R5=%^O5  
R6=%^O6  
R7=%^O7  
SP=%^O6  
PC=%^O7  
.LIST
```


tiny-c/11 Version 11-01-01
TCMAIN

78			.TITLE tiny-c/11	
79			;	
80			; tiny-c/11 interpreter main program	
81			;	
1			.SBTTL MAIN	
2	000000'		.CSECT TCMAIN	
3	000000	010605	TINYC: MOV SP,R5	
4	000002	016706	MOV MSTACK,SP	
5	000006	004567	JSR R5,SAVE	; initialize subroutine call stack
6	000012	016767	MOV BPR,CURSOR	
7	000020	004767	JSR PC,FLOAD	; load tclload.tcf into program space
8	000024	005067	CLR ERR	; initialize tiny-c globals
9	000030	005067	CLR ERRAT	
10	000034	005067	CLR LEAVE	
11	000040	005067	CLR BRAKE	
12	000044	005067	CLR APPLVL	
13	000050	016767	MOV BSTACK,TOP	
14	000056	162767	SUB #6,TOP	
15	000064	016767	MOV BVAR,NXTVAR	
16	000072	016767	MOV BFUN,CURFUN	
17	000100	162767	SUB #6,CURFUN	
18	000106	016767	MOV BFUN,CURGLO	
19	000114	062767	ADD #6,CURGLO	
20	000122	016767	MOV BPR,CURSOR	
21	000130	016767	MOV PROGEND,PRUSED	
22	000136	004767	JSR PC,LINK	; build library level in variable table
23	000142	004767	JSR PC,NEWFUN	
24	000146	016767	MOV BPR,CURSOR	
25	000154	004767	JSR PC,PRBEGN	
26	000160	004767	JSR PC,ST	; start tiny-c interpreter
27	000164	004767	JSR PC,PRDONE	
28	000170	104350	EMT ^O350	; exit to RT-11 monitor

tiny-c/11 Version 11-01-01
TCMAIN

```
29 ;  
30 ; tiny-c/11 literals  
31 ;  
32 000172 164 151 156 VERSION: .ASCII .tiny-c/11 Interpreter Version 11-01-01.  
    000175 171 055 143  
    000200 057 061 061  
    000203 040 111 156  
    000206 164 145 162  
    000211 160 162 145  
    000214 164 145 162  
    000217 040 040 126  
    000222 145 162 163  
    000225 151 157 156  
    000230 040 061 061  
    000233 055 060 061  
    000236 055 060 061  
33 000241 103 157 160 CPYRTE: .ASCII .Copyright [c] 1977 by tiny-c Associates.  
    000244 171 162 151  
    000247 147 150 164  
    000252 040 133 143  
    000255 135 040 061  
    000260 071 067 067  
    000263 040 142 171  
    000266 040 164 151  
    000271 156 171 055  
    000274 143 040 101  
    000277 163 163 157  
    000302 143 151 141  
    000305 164 145 163
```


tiny-c/11 Version 11-01-01
TCMAIN

34	000310	133	000	LBTRACE: .ASCIZ	.[.
35	000312	135	000	RBRACE: .ASCIZ	].
36	000314	050	000	LPAREN: .ASCIZ	.(.
37	000316	051	000	RPAREN: .ASCIZ	)..
38	000320	054	000	COMMA: .ASCIZ	.,.
39	000322	073	000	SEMI: .ASCIZ	;;.
40	000324	012	000	LF: .ASCIZ	<12>
41	000326	115	103	000 MCLIT: .ASCIZ	.MC.
42	000331	145	156	144 ENDLIB: .ASCIZ	.endlibrary.
	000334	154	151	142	
	000337	162	141	162	
	000342	171	000		
43	000344	151	156	164 INT: .ASCIZ	.int.
	000347	000			
44	000350	143	150	141 CHAR: .ASCIZ	.char.
	000353	162	000		
45	000355	151	146	000 IF: .ASCIZ	.if.
46	000360	145	154	163 ELSE: .ASCIZ	.else.
	000363	145	000		
47	000365	167	150	151 WHILE: .ASCIZ	.while.
	000370	154	145	000	
48	000373	142	162	145 BREAK: .ASCIZ	.break.
	000376	141	153	000	
49	000401	162	145	164 RETERN: .ASCIZ	.return.
	000404	165	162	156	
	000407	000			
50	000410	075	000	EQUAL: .ASCIZ	.=.
51	000412	053	000	PLUS: .ASCIZ	.+.
52	000414	055	000	MINUS: .ASCIZ	.-.
53	000416	052	000	TIMES: .ASCIZ	.*.
54	000420	057	000	DIVIDE: .ASCIZ	./.
55	000422	045	000	MODULO: .ASCIZ	.%.
56	000424	075	075	000 EQLEQL: .ASCIZ	==.
57	000427	041	075	000 NOTEQL: .ASCIZ	!=.
58	000432	076	075	000 GRTEQL: .ASCIZ	>=.
59	000435	074	075	000 LESEQ: .ASCIZ	<=.
60	000440	076	000	GRTHN: .ASCIZ	>.
61	000442	074	000	LESTHN: .ASCIZ	<.
62	000001'				.END

tiny-c/11 Version 11-01-01
TCSUBS

```
1          .SBTTL  GLOBALS
2          .NLIST
3          .TITLE  tiny-c/11
4
5          ;
6          ; tiny-c/11 assembly constants
7          ;
8          000036      MAXSTACK=30.
9          000144      MAXVAR=100.
10         000036      MAXFUN=30.
11         047040      MAXPR=20000.
12         000062      MAXDEPTH=50.
13         000010      VLEN=8.
14
15         ;
16         ; tiny-c/11 error codes
17         ;
18         000001      STATERR=1.
19         000002      CURSERR=2.
20         000003      SYMBOLERR=3.
21         000005      RPARERR=5.
22         000006      RANGERR=6.
23         000007      CLASSERR=7.
24         000011      SYNTAXERR=9.
25         000016      LVALERR=14.
26         000017      TOPERR=15.
27         000020      PUSHERR=16.
28         000021      TMFUNERR=17.
29         000022      TMVARERR=18.
30         000023      TMVALERR=19.
31         000024      LINKERR=20.
32         000025      ARGSERR=21.
33         000026      LBRCERR=22.
34         000030      MCERR=24.
35         000031      LOADERR=25.
36         000032      SYMERRA=26.
37
38         ;
39         ; tiny-c/11 global literals
40         ;
41         .GLOBL  LBRACE,RBRACE,LPAREN,RPAREN,COMMA,SEMI,LF,MCLIT,ENDLIB
42         .GLOBL  INT,CHAR,IF,ELSE,WHILE,BREAK,RETERN
43         .GLOBL  EQUAL,PLUS,MINUS,TIMES,DIVIDE,MODULO
44         .GLOBL  EQLEQL,NOTEQL,GRTEQL,LESEQ,GRTHN,LESTHN
45
46         ;
47         ; tiny-c/11 global variables
48         ;
49         .GLOBL  ERR,ERRAT,LEAVE,BRAKE
50         .GLOBL  TOP,NXTVAR,CURFUN,CURGLO,FNAME,LNAME
51         .GLOBL  APPLVL,STCURS,CURSOR,PRUSED,PROGEN
```


tiny-c/11 Version 11-01-01
TCSUBS

```
48  
49 ; tiny-c/11 subroutines  
50 ;  
51 .GLOBL TINYC, ADDRVA, ALPHA, ALPHAN, ASGN, ATOI, BLANKS  
52 .GLOBL CANON, CEQ, CEQN, CONST, ENTER, ESET  
53 .GLOBL EQ, EXPR, FACTOR, FUNDON, LINK, LIT  
54 .GLOBL MC, MOVN, NEWFUN, NEWVAR, NUM, POP, PUSH, RELN, REM  
55 .GLOBL RETURN, SAVE, SETARG, SKIP, SKIPST, ST, SYMNAM  
56 .GLOBL TCTOI, TERM, TOPTOI, VALLOC  
57 ;  
58 ; tiny-c/11 installation vector entries  
59 ;  
60 .GLOBL INCH, OUTCH, FLOAD, FOPEN, FREAD, FWRITE, FCLOSE  
61 .GLOBL USERMC, PRBEGN, STBEGN, PRDONE  
62 .GLOBL BSTACK, ESTACK, BVAR, EVAR, BFUN, EFUN, BPR, EPR, MSTACK  
63 ;  
64 ; tiny-c/11 register assignments  
65 ;  
66 000000 R0=%^00  
67 000001 R1=%^01  
68 000002 R2=%^02  
69 000003 R3=%^03  
70 000004 R4=%^04  
71 000005 R5=%^05  
72 000006 R6=%^06  
73 000007 R7=%^07  
74 000006 SP=%^06  
75 000007 PC=%^07  
76 .LIST
```


tiny-c/11 Version 11-01-01
TCSUBS

```

78                                     .TITLE tiny-c/11
79                                     ;
80                                     ; subroutines comprising the tiny-c interpreter
81                                     ;
82                                     .SBTTL ADDRVAL
83                                     .CSECT TCSUBS
84
85                                     ; find the address in variable table of the symbol between FNAME & LNAME
86                                     ; class of symbol returned at 2(SP), size at 4(SP), length at 6(SP)
87                                     ;
88 000000 004567 006330 ADDRVA: JSR      R5,SAVE
89 000004 162706 000020 SUB        #VLEN+8.,SP
90 000010 016702 000000G MOV        CURFUN,R2
91 000014 010516 MOV        R5,(SP)
92 000016 062716 177762 ADD        #-VLEN-6.,(SP)
93 000022 004737 000652' JSR        PC,@#CANON           ; canonicalize the name
94 000026 005004 CLR        R4
95 000030 011203 1$: MOV        @R2,R3           ; loop over levels: locals, args, globals
96 000032 010200 2$: MOV        R2,R0           ; loop over level's names
97 000034 020360 000002 CMP        R3,2(R0)
98 000040 101047 BHI        5$
99 000042 012716 000010 MOV        #VLEN,(SP)
100 000046 010346 MOV        R3,-(SP)
101 000050 010546 MOV        R5,-(SP)
102 000052 062716 177762 ADD        #-VLEN-6.,(SP)
103 000056 004737 001020' JSR        PC,@#CEQN           ; test if names are equal
104 000062 022626 CMP        (SP)+,(SP)+
105 000064 005700 TST        R0
106 000066 001431 BEQ        4$
107 000070 010300 MOV        R3,R0
108 000072 116075 000010 000004 MOVB       VLEN(R0),@4(R5)           ; store returned values
109 000100 116075 000011 000006 MOVB       VLEN+1.(R0),@6(R5)
110 000106 016075 000012 000010 MOV        VLEN+2.(R0),@10(R5)
111 000114 105775 000004 TSTB       @4(R5)           ; if 0<class<'E', resolve address
112 000120 003410 BLE        3$
113 000122 122775 000105 000004 CMPB       #'E',@4(R5)
114 000130 003404 BLE        3$
115 000132 062700 000014 ADD        #VLEN+4.,R0
116 000136 000167 006154 JMP        RETURN
117 000142 016000 000014 3$: MOV        VLEN+4.(R0),R0
118 000146 000167 006144 JMP        RETURN
119 000152 062703 000016 4$: ADD        #VLEN+6.,R3           ; next name
120 000156 000725 BR        2$
121 000160 005704 5$: TST        R4           ; next level
122 000162 001407 BEQ        6$
123 000164 022704 000001 CMP        #1,R4
124 000170 001010 BNE        7$
125 000172 016702 000000G MOV        BFUN,R2
126 000176 005204 INC        R4
127 000200 000713 BR        1$

```


tiny-c/11 Version 11-01-01
TCSUBS

128	000202	016702	000000G	6\$:	MOV	CURGLO,R2	
129	000206	005204			INC	R4	
130	000210	000707			BR	1\$	
131	000212	012716	000032	7\$:	MOV	#SYMERRA,(SP)	; didn't find it
132	000216	004737	002352'		JSR	PC,@#ESET	; report SYMbol ERRor Alternate
133	000222	005000			CLR	R0	
134	000224	000167	006066		JMP	RETURN	
1					.SBTTL	ALPHA	
2							
3							
4							
5	000230	012700	000001				
6	000234	122766	000141 000002	ALPHA:	MOV	#1,R0	
7	000242	003005			CMPB	#'a,2(SP)	; lower case?
8	000244	122766	000172 000002		BGT	1\$	
9	000252	002412			CMPB	#'z,2(SP)	
10	000254	000207			BLT	2\$	
11	000256	122766	000101 000002	1\$:	RTS	PC	; yes
12	000264	003005			CMPB	#'A,2(SP)	; UPPER CASE?
13	000266	122766	000132 000002		BGT	2\$	
14	000274	002401			CMPB	#'Z,2(SP)	
15	000276	000207			BLT	2\$	
16	000300	005000		2\$:	RTS	PC	; YES
17	000302	000207			CLR	R0	; not alphabetic
1					RTS	PC	
2					.SBTTL	ALPHANUM	
3							
4							
5	000304	012700	000001				
6	000310	122766	000141 000002	ALPHAN:	MOV	#1,R0	
7	000316	003005			CMPB	#'a,2(SP)	; lower case?
8	000320	122766	000172 000002		BGT	1\$	
9	000326	002423			CMPB	#'z,2(SP)	
10	000330	000207			BLT	3\$	
11	000332	122766	000101 000002	1\$:	RTS	PC	; yes
12	000340	003005			CMPB	#'A,2(SP)	; UPPER CASE?
13	000342	122766	000132 000002		BGT	2\$	
14	000350	002412			CMPB	#'Z,2(SP)	
15	000352	000207			BLT	3\$	
16	000354	122766	000060 000002	2\$:	RTS	PC	; YES
17	000362	003005			CMPB	#'0,2(SP)	; numeric?
18	000364	122766	000071 000002		BGT	3\$	
19	000372	002401			CMPB	#'9,2(SP)	
20	000374	000207			BLT	3\$	
21	000376	005000		3\$:	RTS	PC	; yes
22	000400	000207			CLR	R0	; not alphanumeric
					RTS	PC	


```

1
2
3
4
5 000402 004567 005726
6 000406 004767 005034
7 000412 012716 000000G
8 000416 004737 004504'
9 000422 005700
10 000424 001407
11 000426 004767 177750
12 000432 005767 000000G
13 000436 001005
14 000440 004767 001732
15 000444 005767 000000G
16 000450 001403
17 000452 005000
18 000454 000167 005636
19 000460 012700 000001
20 000464 000167 005626

1
2
3
4
5 000470 122776 000040 000002
6 000476 001003
7 000500 005266 000002
8 000504 000771
9 000506 005001
10 000510 005046
11 000512 122776 000055 000004
12 000520 001004
13 000522 012716 000001
14 000526 005266 000004
15 000532 117600 000004
16 000536 162700 000060
17 000542 005700
18 000544 002411
19 000546 022700 000012
20 000552 003406
21 000554 070127 000012
22 000560 060001
23 000562 005266 000004
24 000566 000761
25 000570 005726
26 000572 001401
27 000574 005401
28 000576 010100
29 000600 000207

.SBTTL ASGN
;
; match a relation or an assignment
;
ASGN: JSR R5,SAVE
      JSR PC,RELN ; relation or an lvalue
      MOV #EQUAL,(SP)
      JSR PC,@#LIT ; is there an = sign?
      TST R0
      BEQ 1$
      JSR PC,ASGN ; yes ... match the right side
      TST ERR
      BNE 2$
      JSR PC,EQ ; and make the assignment
1$: TST ERR
   BEQ 3$
2$: CLR R0
   JMP RETURN
3$: MOV #1,R0
4$: JMP RETURN

.SBTTL ATOI
;
; returns value of numeric character string at 2(SP)
;
ATOI: CMPB #' ,@2(SP) ; move by blanks
      BNE 1$
      INC 2(SP)
      BR ATOI
1$: CLR R1
   CLR -(SP)
      CMPB #'-,@4(SP) ; check for minus sign
      BNE 2$
      MOV #1,(SP)
      INC 4(SP)
2$: MOVB @4(SP),R0 ; get next digit
   SUB #'0,R0 ; change to numeric
   TST R0 ; make sure it's a digit
   BLT 3$
   CMP #10.,R0
   BLE 3$
   MUL #10.,R1 ; shift sum
   ADD R0,R1 ; accumulate digit
   INC 4(SP)
   BR 2$
3$: TST (SP)+ ; apply sign
   BEQ 4$
   NEG R1
4$: MOV R1,R0
   RTS PC

```


tiny-c/11 Version 11-01-01
TCSUBS

```

1
2
3
4
5 000602 016700 000000G .SBTTL BLANKS
6 000606 122777 000040 000000G 1$: CMPB #',@CURSOR ; save CURSOR for diagnostic
7 000614 001015 BNE 2$ ; if not a blank, return
8 000616 005267 000000G INC CURSOR
9 000622 026767 000000G 000000G CMP PROGEND,CURSOR ; check for run away
10 000630 103366 BHIS 1$
11 000632 010067 000000G MOV R0,CURSOR ; reset CURSOR to start of run away
12 000636 012746 000002 MOV #CURSERR,-(SP)
13 000642 004737 002352' JSR PC,@#ESET ; and report a CURSOR ERROR
14 000646 005726 TST (SP)+
15 000650 000207 2$: RTS PC
1
2 .SBTTL CANON
3 ; puts canonicalized name between FNAME and LNAME at 2(SP)
4
5 000652 004567 005456 CANON: JSR R5,SAVE
6 000656 016503 000004 MOV 4(R5),R3
7 000662 062703 000007 ADD #VLEN-1,R3
8 000666 016502 000004 MOV 4(R5),R2
9 000672 020302 1$: CMP R3,R2 ; blank out returned name
10 000674 103403 BLO 2$
11 000676 105012 CLRB @R2
12 000700 005202 INC R2
13 000702 000773 BR 1$
14 000704 016502 000004 2$: MOV 4(R5),R2
15 000710 020302 3$: CMP R3,R2
16 000712 103412 BLO 4$
17 000714 117712 000000G MOVB @FNAME,@R2 ; move in VLEN-1 characters
18 000720 005267 000000G INC FNAME
19 000724 026767 000000G 000000G CMP LNAME,FNAME
20 000732 103402 BLO 4$
21 000734 005202 INC R2
22 000736 000764 BR 3$
23 000740 026767 000000G 000000G 4$: CMP LNAME,FNAME
24 000746 103402 BLO 5$
25 000750 117713 000000G MOVB @LNAME,@R3 ; ... and last char if long
26 000754 000167 005336 5$: JMP RETURN

```


tiny-c/11 Version 11-01-01
TCSUBS

```

1                                     .SBTTL CEQ
2                                     ;
3                                     ; test char string at 2(SP) matches string at 4(SP)
4                                     ;
5 000760 005000 CEQ: CLR R0
6 000762 105776 000002 1$: TSTB @2(SP)
7 000766 001411 BEQ 2$
8 000770 127676 000002 000004 CMPB @2(SP),@4(SP)
9 000776 001007 BNE 3$
10 001000 005266 000002 INC 2(SP)
11 001004 005266 000004 INC 4(SP)
12 001010 000764 BR 1$
13 001012 012700 000001 2$: MOV #1,R0
14 001016 000207 3$: RTS PC
                                     .SBTTL CEQN
1                                     ;
2                                     ; test 6(SP) bytes at 2(SP) match bytes at 4(SP)
3                                     ;
4                                     ;
5 001020 005000 CEQN: CLR R0
6 001022 005766 000006 TST 6(SP)
7 001026 003413 BLE 2$
8 001030 127676 000002 000004 1$: CMPB @2(SP),@4(SP)
9 001036 001011 BNE 3$
10 001040 005266 000002 INC 2(SP)
11 001044 005266 000004 INC 4(SP)
12 001050 005366 000006 DEC 6(SP)
13 001054 003365 BGT 1$
14 001056 012700 000001 2$: MOV #1,R0
15 001062 000207 3$: RTS PC
                                     .SBTTL CONST
1                                     ;
2                                     ; match constants ... numbers, strings, & characters
3                                     ; set FNAME & LNAME around it if it matches
4                                     ;
5                                     ;
6 001064 004567 005244 CONST: JSR R5,SAVE
7 001070 016702 000000G MOV CURSOR,R2 ; save CURSOR for diagnostic
8 001074 004767 177502 JSR PC,BLANKS ; clear away blanks
9 001100 122777 000053 000000G CMPB #'@CURSOR ; is it a number?
10 001106 001414 BEQ 1$
11 001110 122777 000055 000000G CMPB #'@CURSOR
12 001116 001410 BEQ 1$
13 001120 117700 000000G MOVB @CURSOR,R0
14 001124 010016 MOV R0,(SP)
15 001126 004737 005276' JSR PC,@#NUM
16 001132 022700 177777 CMP #-1,R0
17 001136 001427 BEQ 4$
18 001140 016767 000000G 000000G 1$: MOV CURSOR,FNAME
19 001146 005267 000000G 2$: INC CURSOR

```


tiny-c/11 Version 11-01-01
TCSUBS

20	001152	117700	000000G	MOVB	@CURSOR,R0	
21	001156	010016		MOV	R0,(SP)	
22	001160	004737	005276'	JSR	PC,@#NUM	
23	001164	022700	177777	CMP	#-1,R0	
24	001170	001401		BEQ	3\$	
25	001172	000765		BR	2\$	
26	001174	016767	000000G 000000G 3\$:	MOV	CURSOR,LNAME	
27	001202	005367	000000G	DEC	LNAME	
28	001206	012700	000001	MOV	#1,R0	
29	001212	000167	005100	JMP	RETURN	
30	001216	122777	000042 000000G 4\$:	CMPB	#'",@CURSOR	; is it a string?
31	001224	001050		BNE	7\$	
32	001226	005267	000000G	INC	CURSOR	
33	001232	016767	000000G 000000G	MOV	CURSOR,FNAME	
34	001240	122777	000042 000000G 5\$:	CMPB	#'",@CURSOR	
35	001246	001421		BEQ	6\$	
36	001250	105777	000000G	TSTB	@CURSOR	
37	001254	001416		BEQ	6\$	
38	001256	005267	000000G	INC	CURSOR	
39	001262	026767	000000G 000000G	CMP	PROGEND,CURSOR	; check for run away
40	001270	103363		BHIS	5\$	
41	001272	010267	000000G	MOV	R2,CURSOR	; reset CURSOR to start of run away
42	001276	012716	000002	MOV	#CURSERR,(SP)	
43	001302	004737	002352'	JSR	PC,@#ESET	; and report a CURSOR error
44	001306	000167	005004	JMP	RETURN	
45	001312	105077	000000G 6\$:	CLRB	@CURSOR	; make it a tiny-c string
46	001316	005267	000000G	INC	CURSOR	
47	001322	016767	000000G 000000G	MOV	CURSOR,LNAME	
48	001330	162767	000002 000000G	SUB	#2,LNAME	
49	001336	012700	000002	MOV	#2,R0	
50	001342	000167	004750	JMP	RETURN	
51	001346	122777	000047 000000G 7\$:	CMPB	#'",@CURSOR	; is it a character?
52	001354	001024		BNE	8\$	
53	001356	005267	000000G	INC	CURSOR	
54	001362	016767	000000G 000000G	MOV	CURSOR,FNAME	
55	001370	012716	000047	MOV	#'',(SP)	
56	001374	005046		CLR	-(SP)	
57	001376	004737	006456'	JSR	PC,@#SKIP	
58	001402	016767	000000G 000000G	MOV	CURSOR,LNAME	
59	001410	162767	000002 000000G	SUB	#2,LNAME	
60	001416	012700	000003	MOV	#3,R0	
61	001422	000167	004670	JMP	RETURN	
62	001426	005000		CLR	R0	; not a constant
63	001430	000167	004662	JMP	RETURN	

tiny-c/11 Version 11-01-01
TCSUBS

```

1                                     .SBTTL  ENTER
2                                     ;
3                                     ; enter a called function ... assign arguments to locals
4                                     ;
5 001434 004567 004674 ENTER: JSR    R5,SAVE
6 001440 162706 000012 SUB     #12,SP
7 001444 016765 000000G 177762 MOV   CURSOR,-16(R5) ; save CURSOR for diagnostic
8 001452 005065 177766 CLR     -12(R5) ; set argument count to 0
9 001456 016700 000000G MOV   TOP,R0
10 001462 062700 000006 ADD     #6,R0
11 001466 010065 177764 MOV   R0,-14(R5)
12 001472 012716 000000G MOV   #LPAREN,(SP) ; match (optional) left paren
13 001476 004737 004504' JSR   PC,@#LIT
14 001502 012716 000000G MOV   #RPAREN,(SP) ; test for no arguments
15 001506 004737 004504' JSR   PC,@#LIT
16 001512 005700 TST     R0
17 001514 001067 BNE     6$
18 001516 122777 000135 000000G CMPB  #'],@CURSOR
19 001524 001463 BEQ     6$
20 001526 122777 000073 000000G CMPB  #'',@CURSOR
21 001534 001457 BEQ     6$
22 001536 122777 000012 000000G CMPB  #12,@CURSOR
23 001544 001453 BEQ     6$
24 001546 122777 000057 000000G CMPB  #'/',@CURSOR
25 001554 001447 BEQ     6$
26 001556 005767 000000G 1$: TST    ERR
27 001562 001012 BNE     2$
28 001564 004767 176612 JSR   PC,ASGN ; stack the argument values
29 001570 005265 177766 INC     -12(R5) ; while counting them
30 001574 012716 000000G MOV   #COMMA,(SP)
31 001600 004737 004504' JSR   PC,@#LIT
32 001604 005700 TST     R0
33 001606 001363 BNE     1$
34 001610 005767 000000G 2$: TST    ERR
35 001614 001423 BEQ     5$
36 001616 005765 177766 TST    -12(R5) ; if ASGN trouble, pop all values but 1
37 001622 001012 BNE     4$
38 001624 005016 CLR     (SP) ; if no values where pushed, push a 0
39 001626 012746 000002 MOV   #2,-(SP)
40 001632 012746 000101 MOV   #'A,-(SP)
41 001636 005046 CLR     -(SP)
42 001640 004737 005356' JSR   PC,@#PUSH
43 001644 000167 004446 3$: JMP    RETURN
44 001650 005365 177766 4$: DEC    -12(R5)
45 001654 003773 BLE     3$
46 001656 004767 003454 JSR   PC,POP
47 001662 000772 BR      4$
48 001664 012716 000000G 5$: MOV   #RPAREN,(SP) ; match the right paren

```


tiny-c/11 Version 11-01-01
TCSUBS

49	001670	004737	004504'		JSR	PC,@#LIT	
50	001674	005765	000004	6\$:	TST	4(R5)	; if "branch" address is 0, call MC
51	001700	001006		ENTER1:	BNE	7\$	
52	001702	016516	177766		MOV	-12(R5),(SP)	; with the no. of arguments
53	001706	004737	000000G		JSR	PC,@#MC	
54	001712	000167	004400		JMP	RETURN	
55	001716	016765	000000G	177770	7\$:	MOV	CURSOR,-10(R5)
56	001724	016765	000000G	177760		MOV	STCURS,-20(R5)
57	001732	016567	000004	000000G		MOV	4(R5),CURSOR
58	001740	004767	002710		JSR	PC,NEWFUN	; and "branch"
59	001744	004767	004226	8\$:	JSR	PC,REM	; push function on function table
60	001750	012716	000000G		MOV	#INT,(SP)	; match int statement
61	001754	004737	004504'		JSR	PC,@#LIT	
62	001760	005700			TST	R0	
63	001762	001440			BEQ	11\$	
64	001764	012716	000111		MOV	#'I,(SP)	
65	001770	016546	177764		MOV	-14(R5),-(SP)	
66	001774	062765		177764	ADD	#6,-14(R5)	
67	002002	004737	006350'		JSR	PC,@#SETARG	; and set their values
68	002006	005726			TST	(SP)+	
69	002010	012716	000000G	9\$:	MOV	#COMMA,(SP)	
70	002014	004737	004504'		JSR	PC,@#LIT	
71	002020	005700			TST	R0	
72	002022	001413			BEQ	10\$	
73	002024	012716	000111		MOV	#'I,(SP)	
74	002030	016546	177764		MOV	-14(R5),-(SP)	
75	002034	062765		177764	ADD	#6,-14(R5)	
76	002042	004737	006350'		JSR	PC,@#SETARG	
77	002046	005726			TST	(SP)+	
78	002050	000757			BR	9\$	
79	002052	012716	000000G	10\$:	MOV	#SEMI,(SP)	
80	002056	004737	004504'		JSR	PC,@#LIT	
81	002062	000730			BR	8\$	
82	002064	012716	000000G	11\$:	MOV	#CHAR,(SP)	; match char statement
83	002070	004737	004504'		JSR	PC,@#LIT	
84	002074	005700			TST	R0	
85	002076	001440			BEQ	14\$	
86	002100	012716	000103		MOV	#'C,(SP)	
87	002104	016546	177764		MOV	-14(R5),-(SP)	
88	002110	062765		177764	ADD	#6,-14(R5)	
89	002116	004737	006350'		JSR	PC,@#SETARG	; and set their values
90	002122	005726			TST	(SP)+	
91	002124	012716	000000G	12\$:	MOV	#COMMA,(SP)	
92	002130	004737	004504'		JSR	PC,@#LIT	
93	002134	005700			TST	R0	
94	002136	001413			BEQ	13\$	
95	002140	012716	000103		MOV	#'C,(SP)	
96	002144	016546	177764		MOV	-14(R5),-(SP)	

tiny-c/11 Version 11-01-01
TCSUBS

```

97 002150 062765 000006 177764      ADD    #6,-14(R5)
98 002156 004737 006350'          JSR    PC,@#SETARG
99 002162 005726                    TST    (SP)+
100 002164 000757                    BR     12$
101 002166 012716 000000G          13$: MOV    #SEMI,(SP)
102 002172 004737 004504'          JSR    PC,@#LIT
103 002176 000662                    BR     8$
104 002200 016700 000000G          14$: MOV    TOP,R0
105 002204 062700 000006          ADD    #6,R0
106 002210 026500 177764          CMP    -14(R5),R0      ; test if no. of arguments correct
107 002214 001407                    BEQ    15$
108 002216 012716 000025          MOV    #ARGSERR,(SP)      ; if not, raise an ARGS error
109 002222 016567 177762 000000G  MOV    -16(R5),CURSOR      ; restore function CURSOR for diagnostic
110 002230 004737 002352'          JSR    PC,@#ESET
111 002234 016500 177766          15$: MOV    -12(R5),R0      ; POP off the argument values
112 002240 005365 177766          DEC    -12(R5)
113 002244 005700                    TST    R0
114 002246 003403                    BLE    16$
115 002250 004767 003062          JSR    PC,POP
116 002254 000767                    BR     15$
117 002256 005767 000000G          16$: TST    ERR
118 002262 001002                    BNE    17$
119 002264 004767 004572          JSR    PC,ST      ; if no ERROR, start interpreter in function
120 002270 005767 000000G          17$: TST    LEAVE
121 002274 001012          ENTER2: BNE    18$
122 002276 005016                    CLR    (SP)      ; if LEAVE flag not set, push a 0
123 002300 012746 000002          MOV    #2,-(SP)
124 002304 012746 000101          MOV    #'A,-(SP)
125 002310 005046                    CLR    -(SP)
126 002312 004737 005356'          JSR    PC,@#PUSH
127 002316 062706 000006          ADD    #6,SP
128 002322 005067 000000G          18$: CLR    LEAVE
129 002326 016567 177760 000000G  MOV    -20(R5),STCURS      ; restore "return" addresses
130 002334 016567 177770 000000G  MOV    -10(R5),CURSOR
131 002342 004767 001444          JSR    PC,FUNDONE      ; pop the function table
132 002346 000167 003744          JMP    RETURN
1          .SBTTL  ESET
2
3          ; error trap routine
4          ;
5 002352 005767 000000G          ESET: TST    ERR
6 002356 001006                    BNE    1$
7 002360 016667 000002 000000G  MOV    2(SP),ERR      ; set error cell if not already set
8 002366 016767 000000G 000000G  MOV    CURSOR,ERRAT      ; & capture current CURSOR
9 002374 000207          1$: RTS    PC

```


tiny-c/11 Version 11-01-01
TCSUBS

1			.SBTTL EQ	
2			;	
3			; perform an assignment	
4			;	
5	002376	004567	EQ: JSR	R5,SAVE
6	002402	162706	SUB	#4,SP
7	002406	016700	MOV	TOP,R0
8	002412	122760	CMPB	#'L,-5(R0)
		000114		; make sure top of stack is an lvalue
9	002420	001055	BNE	3\$
10	002422	004767	JSR	PC,TOPTOI
		006120		; get the value
11	002426	010065	MOV	R0,-10(R5)
12	002432	004767	JSR	PC,POP
		002700		; & where to put it
13	002436	010065	MOV	R0,-12(R5)
14	002442	016700	MOV	TOP,R0
15	002446	105760	TSBTB	6(R0)
		000006		; if class=0, move "size" bytes
16	002452	001014	BNE	1\$
17	002454	116000	MOVB	10(R0),R0
18	002460	010016	MOV	R0,(SP)
19	002462	010546	MOV	R5,-(SP)
20	002464	062716	ADD	#-10,(SP)
21	002470	016546	MOV	-12(R5),-(SP)
22	002474	004737	JSR	PC,@#MOVN
23	002500	022626	CMP	(SP)+,(SP)+
24	002502	000412	BR	2\$
25	002504	012716	1\$: MOV	#2,(SP)
		000002		; otherwise move 2 bytes
26	002510	010546	MOV	R5,-(SP)
27	002512	062716	ADD	#-10,(SP)
28	002516	016546	MOV	-12(R5),-(SP)
29	002522	004737	JSR	PC,@#MOVN
30	002526	022626	CMP	(SP)+,(SP)+
31	002530	016516	2\$: MOV	-10(R5),(SP)
		177770		; push the value
32	002534	012746	MOV	#2,-(SP)
33	002540	012746	MOV	#'A,-(SP)
34	002544	005046	CLR	-(SP)
35	002546	004737	JSR	PC,@#PUSH
36	002552	000414	BR	4\$
37	002554	012716	3\$: MOV	#LVALERR,(SP)
		000016		; TOP not an lvalue
38	002560	004737	JSR	PC,@#ESET
		002352		; report an LVALue ERRor
39	002564	005016	CLR	(SP)
		000002		; push a 0
40	002566	012746	MOV	#2,-(SP)
41	002572	012746	MOV	#'A,-(SP)
42	002576	005046	CLR	-(SP)
43	002600	004737	JSR	PC,@#PUSH
44	002604	000167	4\$: JMP	RETURN
		003506		


```

1
2
3
4
5 002610 004567 003520
6 002614 012716 000000G
7 002620 004737 004504'
8 002624 005700
9 002626 001420
10 002630 004767 005406
11 002634 004767 005706
12 002640 005400
13 002642 010016
14 002644 012746 000002
15 002650 012746 000101
16 002654 005046
17 002656 004737 005356'
18 002662 062706 000006
19 002666 000406
20 002670 012716 000000G
21 002674 004737 004504'
22 002700 004767 005336
23 002704 005767 000000G
24 002710 001061
25 002712 012716 000000G
26 002716 004737 004504'
27 002722 005700
28 002724 001422
29 002726 004767 005310
30 002732 004767 005610
31 002736 010016
32 002740 004767 005602
33 002744 060016
34 002746 012746 000002
35 002752 012746 000101
36 002756 005046
37 002760 004737 005356'
38 002764 062706 000006
39 002770 000745
40 002772 012716 000000G
41 002776 004737 004504'
42 003002 005700
43 003004 001423
44 003006 004767 005230
45 003012 004767 005530
46 003016 005400

```

```

.SBTTL  EXPR
;
; an EXPression is a term or sum of terms
;
EXPR:  JSR      R5,SAVE
        MOV      #MINUS,(SP)      ; test for unary minus
        JSR      PC,@#LIT
        TST      R0
        BEQ      1$
        JSR      PC,TERM          ; evaluate a term
        JSR      PC,TOPTOI
        NEG      R0              ; & push its negative
        MOV      R0,(SP)
        MOV      #2,-(SP)
        MOV      #'A,-(SP)
        CLR      -(SP)
        JSR      PC,@#PUSH
        ADD      #6,SP
        BR       2$
1$:     MOV      #PLUS,(SP)      ; test for unary plus
        JSR      PC,@#LIT
        JSR      PC,TERM          ; evaluate a term & leave it on the stack
2$:     TST      ERR
        BNE      4$
        MOV      #PLUS,(SP)      ; test for sum of terms
        JSR      PC,@#LIT
        TST      R0
        BEQ      3$
        JSR      PC,TERM          ; evaluate the addend
        JSR      PC,TOPTOI
        MOV      R0,(SP)
        JSR      PC,TOPTOI        ; get the augend
        ADD      R0,(SP)          ; form sum & push it
        MOV      #2,-(SP)
        MOV      #'A,-(SP)
        CLR      -(SP)
        JSR      PC,@#PUSH
        ADD      #6,SP
        BR       2$
3$:     MOV      #MINUS,(SP)      ; test for difference of terms
        JSR      PC,@#LIT
        TST      R0
        BEQ      4$
        JSR      PC,TERM          ; evaluate the subtrahend
        JSR      PC,TOPTOI
        NEG      R0              ; and negate it

```


tiny-c/11 Version 11-01-01
TCSUBS

47	003020	010016		MOV	R0,(SP)	
48	003022	004767	005520	JSR	PC,TOPTOI	; get the minuend
49	003026	060016		ADD	R0,(SP)	; form difference & push it
50	003030	012746	000002	MOV	#2,-(SP)	
51	003034	012746	000101	MOV	#'A,-(SP)	
52	003040	005046		CLR	-(SP)	
53	003042	004737	005356'	JSR	PC,@#PUSH	
54	003046	062706	000006	ADD	#6,SP	
55	003052	000714		BR	2\$	
56	003054	000167	003236	4\$: JMP	RETURN	
1					.SBTTL	FACTOR
2						
3						
4						
5						
6						
7						
8						
9	003060	004567	003250	FACTOR: JSR	R5,SAVE	
10	003064	162706	000014	SUB	#14,SP	
11	003070	012716	000000G	MOV	#LPAREN,(SP)	; check for (assignment)
12	003074	004737	004504'	JSR	PC,@#LIT	
13	003100	005700		TST	R0	
14	003102	001420		BEQ	2\$	
15	003104	004767	175272	JSR	PC,ASGN	
16	003110	012716	000000G	MOV	#RPAREN,(SP)	; match the right paren
17	003114	004737	004504'	JSR	PC,@#LIT	
18	003120	005700		TST	R0	
19	003122	001402		BEQ	1\$	
20	003124	000167	003166	JMP	RETURN	
21	003130	012716	000005	1\$: MOV	#RPARERR,(SP)	; no right paren
22	003134	004737	002352'	JSR	PC,@#ESET	; report Right PAREn ERRor
23	003140	000167	003152	JMP	RETURN	
24	003144	004767	175714	2\$: JSR	PC,CONST	; check for a constant
25	003150	010065	177770	MOV	R0,-10(R5)	
26	003154	001464		BEQ	6\$	
27	003156	022765	000001 177770	CMP	#1,-10(R5)	; constant type=1
28	003164	001016		BNE	3\$	
29	003166	016716	000000G	MOV	FNAME,(SP)	
30	003172	004737	000470'	JSR	PC,@#ATOI	; convert ascii to integer
31	003176	010016		MOV	R0,(SP)	
32	003200	012746	000002	MOV	#2,-(SP)	
33	003204	012746	000101	MOV	#'A,-(SP)	
34	003210	005046		CLR	-(SP)	
35	003212	004737	005356'	JSR	PC,@#PUSH	; and push the integer value
36	003216	000167	003074	JMP	RETURN	
37	003222	022765	000002 177770	3\$: CMP	#2,-10(R5)	; constant type=2
38	003230	001014		BNE	4\$	

tiny-c/11 Version 11-01-01
TCSUBS

39	003232	016716	000000G		MOV	FNAME, (SP)	
40	003236	012746	0000001		MOV	#1, -(SP)	
41	003242	012746	000101		MOV	#'A, -(SP)	
42	003246	012746	0000001		MOV	#1, -(SP)	
43	003252	004737	005356'		JSR	PC, @#PUSH	; push name pointer
44	003256	000167	003034		JMP	RETURN	
45	003262	022765	0000003	177770	4\$: CMP	#3, -10(R5)	; constant type=3
46	003270	001402			BEQ	5\$	
47	003272	000167	003020		JMP	RETURN	
48	003276	117700	000000G	5\$:	MOVB	@FNAME, R0	
49	003302	010016			MOV	R0, (SP)	
50	003304	012746	0000001		MOV	#1, -(SP)	
51	003310	012746	000101		MOV	#'A, -(SP)	
52	003314	005046			CLR	-(SP)	
53	003316	004737	005356'		JSR	PC, @#PUSH	; push pointer to pointer to name
54	003322	000167	002770		JMP	RETURN	
55	003326	004767	004562	6\$:	JSR	PC, SYMNAME	; check for variable reference
56	003332	005700		FACT1:	TST	R0	; or function call
57	003334	001002			BNE	7\$	
58	003336	000167	000414		JMP	FACT3	
59	003342	016700	000000G	7\$:	MOV	LNAME, R0	; is it an MC call?
60	003346	166700	000000G		SUB	FNAME, R0	
61	003352	022700	0000001		CMP	#1, R0	
62	003356	001020			BNE	8\$	
63	003360	012716	0000002		MOV	#2, (SP)	
64	003364	012746	000000G		MOV	#MCLIT, -(SP)	
65	003370	016746	000000G		MOV	FNAME, -(SP)	
66	003374	004737	001020'		JSR	PC, @#CEQN	
67	003400	022626			CMP	(SP) +, (SP) +	
68	003402	005700			TST	R0	
69	003404	001405			BEQ	8\$	
70	003406	005016			CLR	(SP)	
71	003410	004737	001434'		JSR	PC, @#ENTER	; ENTER the MC call
72	003414	000167	002676		JMP	RETURN	
73	003420	010516		8\$:	MOV	R5, (SP)	; function or variable?
74	003422	062716	177766		ADD	#-12, (SP)	
75	003426	010546			MOV	R5, -(SP)	
76	003430	062716	177756		ADD	#-22, (SP)	
77	003434	010546			MOV	R5, -(SP)	
78	003436	062716	177760		ADD	#-20, (SP)	
79	003442	004737	000000'		JSR	PC, @#ADDRVAL	
80	003446	022626			CMP	(SP) +, (SP) +	
81	003450	010065	177762		MOV	R0, -16(R5)	
82	003454	001533		FACT2:	BEQ	14\$	
83	003456	122765	000105	177760	CMPB	#'E, -20(R5)	
84	003464	003515			BLE	13\$	
85	003466	012716	000000G		MOV	#LPAREN, (SP)	; is the variable dimensioned?

tiny-c/11 Version 11-01-01
TCSUBS

86	003472	004737	004504'		JSR	PC,@#LIT		
87	003476	005700			TST	R0		
88	003500	001471			BEQ	12\$		
89	003502	105365	177760		DECB	-20(R5)		
90	003506	105765	177760		TSTB	-20(R5)		
91	003512	002005			BGE	9\$		
92	003514	012716	000007		MOV	#CLASSERR,(SP)	; yes but it shouldn't be	
93	003520	004737	002352'		JSR	PC,@#ESET	; report a CLASS ERRor	
94	003524	000520			BR	FACT4	; and push a 0	
95	003526	012716	000002	9\$:	MOV	#2,(SP)	; evaluate the dimension	
96	003532	016546	177762		MOV	-16(R5),-(SP)		
97	003536	010546			MOV	R5, -(SP)		
98	003540	062716	177762		ADD	#-16,(SP)		
99	003544	004737	004620'		JSR	PC,@#MOVN		
100	003550	022626			CMP	(SP)+,(SP)+		
101	003552	004767	174624		JSR	PC,ASGN		
102	003556	005700			TST	R0		
103	003560	001512			BEQ	FACT5		
104	003562	012716	000000G		MOV	#RPAREN,(SP)		
105	003566	004737	004504'		JSR	PC,@#LIT		
106	003572	004767	004750		JSR	PC,TOPTOI		
107	003576	010065	177764		MOV	R0,-14(R5)		
108	003602	022765	000001	177766	CMP	#1,-12(R5)		
109	003610	002017			BGE	11\$		
110	003612	105765	177760		TSTB	-20(R5)		
111	003616	001014			BNE	11\$		
112	003620	005765	177764		TST	-14(R5)		
113	003624	002404			BLT	10\$		
114	003626	026565	177766	177764	CMP	-12(R5),-14(R5)		
115	003634	003005			BGT	11\$		
116	003636	012716	000006	10\$:	MOV	#RANGERR,(SP)	; if dimension is out of bounds	
117	003642	004737	002352'		JSR	PC,@#ESET	; report a RANGE ERRor	
118	003646	000447			BR	FACT4	; and push a 0	
119	003650	116501	177756	11\$:	MOVB	-22(R5),R1		
120	003654	070165	177764		MUL	-14(R5),R1		
121	003660	060165	177762		ADD	R1,-16(R5)		
122	003664	016516	177762	12\$:	MOV	-16(R5),(SP)	; push the value of the variable	
123	003670	116500	177756		MOVB	-22(R5),R0		
124	003674	010046			MOV	R0, -(SP)		
125	003676	012746	000114		MOV	#'L, -(SP)		
126	003702	116500	177760		MOVB	-20(R5),R0		
127	003706	010046			MOV	R0, -(SP)		
128	003710	004737	005356'		JSR	PC,@#PUSH		
129	003714	000167	002376		JMP	RETURN		
130	003720	122765	000105	177760	13\$:	CMPB	#'E,-20(R5)	; is it a function?
131	003726	001027			BNE	FACT5		
132	003730	016516	177762		MOV	-16(R5),(SP)		

tiny-c/11 Version 11-01-01
TCSUBS

133	003734	004737	001434'	JSR	PC,@#ENTER	; yes ... ENTER the function
134	003740	000167	002352	JMP	RETURN	
135	003744	012716	000003	14\$: MOV	#SYMBOLERR,(SP)	; it's not a known SYMBOL
136	003750	004737	002352'	JSR	PC,@#ESET	; report a SYMBOL ERRor
137	003754	000404		BR	FACT4	; and push a 0
138	003756	012716	000011	FACT3: MOV	#SYNTAXERR,(SP)	; it's not a FACTOR
139	003762	004737	002352'	JSR	PC,@#ESET	; report a SYNTAX ERRor
140	003766	005016		FACT4: CLR	(SP)	; and push a 0
141	003770	012746	000002	MOV	#2,-(SP)	
142	003774	012746	000101	MOV	#'A,-(SP)	
143	004000	005046		CLR	-(SP)	
144	004002	004737	005356'	JSR	PC,@#PUSH	
145	004006	000167	002304	FACT5: JMP	RETURN	
1				.SBTTL	FUNDONE	
2				:		
3				:		; delete the most recent entry in the function table
4				:		
5	004012	017767	000000G 000000G	FUNDON: MOV	@CURFUN,NXTVAR	
6	004020	016700	000000G	MOV	CURFUN,R0	
7	004024	016067	0000004 000000G	MOV	4(R0),PRUSED	
8	004032	162767	0000006 000000G	SUB	#6,CURFUN	
9	004040	000207		RTS	PC	
1				.SBTTL	LINK	
2				:		
3				:		; allocate all externals including all function names
4				:		
5	004042	004567	002266	LINK: JSR	R5,SAVE	
6	004046	004767	000602	JSR	PC,NEWFUN	; establish level 0 in the function table
7	004052	016700	000000G	LINK1: MOV	PROGEND,R0	
8	004056	005300		DEC	R0	
9	004060	026700	000000G	CMP	CURSOR,R0	
10	004064	103001		BHIS	1\$	
11	004066	000402		BR	2\$	
12	004070	000167	002222	1\$: JMP	RETURN	
13	004074	005767	000000G	2\$: TST	ERR	
14	004100	001373		BNE	1\$	
15	004102	004767	002070	JSR	PC,REM	
16	004106	012716	000000G	MOV	#LBRACE,(SP)	; skip over a function's body
17	004112	004737	004504'	JSR	PC,@#LIT	
18	004116	005700		TST	R0	
19	004120	001410		BEQ	3\$	
20	004122	012716	000135	MOV	#'],(SP)	
21	004126	012746	000133	MOV	#'[-(SP)	
22	004132	004737	006456'	JSR	PC,@#SKIP	
23	004136	005726		TST	(SP)+	
24	004140	000744		BR	LINK1	

tiny-c/11 Version 11-01-01
TCSUBS

25 004142 012716 000000G	3\$: MOV	#CHAR, (SP)	; allocate global chars
26 004146 004737 004504'	JSR	PC, @#LIT	
27 004152 005700	TST	R0	
28 004154 001430	BEQ	6\$	
29 004156 005016	CLR	(SP)	
30 004160 012746 000103	MOV	#'C, - (SP)	
31 004164 004737 010700'	JSR	PC, @#VALLOC	
32 004170 005726	TST	(SP) +	
33 004172 012716 000000G	4\$: MOV	#COMMA, (SP)	
34 004176 004737 004504'	JSR	PC, @#LIT	
35 004202 005700	TST	R0	
36 004204 001407	BEQ	5\$	
37 004206 005016	CLR	(SP)	
38 004210 012746 000103	MOV	#'C, - (SP)	
39 004214 004737 010700'	JSR	PC, @#VALLOC	
40 004220 005726	TST	(SP) +	
41 004222 000763	BR	4\$	
42 004224 012716 000000G	5\$: MOV	#SEMI, (SP)	
43 004230 004737 004504'	JSR	PC, @#LIT	
44 004234 000706	BR	LINK1	
45 004236 012716 000000G	6\$: MOV	#INT, (SP)	; allocate global ints
46 004242 004737 004504'	JSR	PC, @#LIT	
47 004246 005700	TST	R0	
48 004250 001430	LINK2: BEQ	9\$	
49 004252 005016	CLR	(SP)	
50 004254 012746 000111	MOV	#'I, - (SP)	
51 004260 004737 010700'	JSR	PC, @#VALLOC	
52 004264 005726	TST	(SP) +	
53 004266 012716 000000G	7\$: MOV	#COMMA, (SP)	
54 004272 004737 004504'	JSR	PC, @#LIT	
55 004276 005700	TST	R0	
56 004300 001407	BEQ	8\$	
57 004302 005016	CLR	(SP)	
58 004304 012746 000111	MOV	#'I, - (SP)	
59 004310 004737 010700'	JSR	PC, @#VALLOC	
60 004314 005726	TST	(SP) +	
61 004316 000763	BR	7\$	
62 004320 012716 000000G	8\$: MOV	#SEMI, (SP)	
63 004324 004737 004504'	JSR	PC, @#LIT	
64 004330 000650	BR	LINK1	
65 004332 012716 000000G	9\$: MOV	#ENDLIB, (SP)	; if end of library
66 004336 004737 004504'	JSR	PC, @#LIT	
67 004342 005700	TST	R0	
68 004344 001403	BEQ	10\$	
69 004346 004767 000302	JSR	PC, NEWFUN	; establish level 1 in the function table
70 004352 000637	BR	LINK1	
71 004354 004767 003534	10\$: JSR	PC, SYMNAME	; allocate a function name
72 004360 005700	TST	R0	

tiny-c/11 Version 11-01-01
TCSUBS

73	004362	001442		BEQ	13\$	
74	004364	016716	000000G	MOV	CURSOR, (SP)	
75	004370	012746	0000001	MOV	#1, -(SP)	
76	004374	012746	0000002	MOV	#2, -(SP)	
77	004400	012746	000105	MOV	#'E, -(SP)	
78	004404	004737	004746'	JSR	PC, @#NEWVAR	
79	004410	062706	0000006	ADD	#6, SP	
80	004414	122777	000133 000000G 11\$:	CMPB	#'[, @CURSOR	; and look for its [
81	004422	001416		BEQ	12\$	
82	004424	005767	000000G	TST	ERR	
83	004430	001013		BNE	12\$	
84	004432	005267	000000G	INC	CURSOR	
85	004436	026767	000000G 000000G	CMP	PROGEND, CURSOR	
86	004444	101363		BHI	11\$	
87	004446	012716	000026	MOV	#LBRCErr, (SP)	; if CURSOR run away,
88	004452	004737	002352'	JSR	PC, @#ESET	; report a Left BRaCe ERror
89	004456	000756		BR	11\$	
90	004460	004767	002072 12\$:	JSR	PC, SKIPST	; skip over function's body
91	004464	000167	177362	JMP	LINK1	
92	004470	012716	000024 13\$:	MOV	#LINKERR, (SP)	; if global syntax trouble,
93	004474	004737	002352'	JSR	PC, @#ESET	; report a LINK ERror
94	004500	000167	177346	JMP	LINK1	
1				.SBTTL	LIT	
2						
3						
4						
5						
6	004504	016701	000000G	LIT: MOV	CURSOR, R1	; save CURSOR for diagnostic
7	004510	122777	000040 000000G	CMPB	#'[, @CURSOR	; bypass blanks
8	004516	001007		BNE	1\$	
9	004520	026767	000000G 000000G	CMP	CURSOR, PROGEND	; check for run away
10	004526	103021		BHIS	3\$	
11	004530	005267	000000G	INC	CURSOR	
12	004534	000763		BR	LIT	
13	004536	016600	0000002 1\$:	MOV	2 (SP), R0	
14	004542	121077	000000G 2\$:	CMPB	(R0), @CURSOR	; match LITeral
15	004546	001020		BNE	4\$	
16	004550	005267	000000G	INC	CURSOR	
17	004554	005200		INC	R0	
18	004556	105710		TSTB	(R0)	; check for end of LITeral string
19	004560	001416		BEQ	5\$	
20	004562	026767	000000G 000000G	CMP	CURSOR, PROGEND	; check for runaway
21	004570	101764		BLOS	2\$	
22	004572	010167	000000G 3\$:	MOV	R1, CURSOR	; restore CURSOR for diagnostic
23	004576	012746	0000002	MOV	#CURSERR, -(SP)	
24	004602	004737	002352'	JSR	PC, @#ESET	; report a CURSor ERror
25	004606	005726		TST	(SP) +	
26	004610	010167	000000G 4\$:	MOV	R1, CURSOR	; reset CURSOR
27	004614	005000		CLR	R0	
28	004616	000207	5\$:	RTS	PC	

tiny-c/11 Version 11-01-01
TCSUBS

```

1                                     .SBTTL  MOVN
2                                     ;
3                                     ; move 6(sp) bytes from 4(sp) to 2(sp)
4                                     ;
5 004620 005766 000006 MOVN:  TST      6(SP)
6 004624 003412          BLE      2$
7 004626 117676 000004 000002 1$:  MOVB    @4(SP),@2(SP)
8 004634 005266 000002          INC      2(SP)
9 004640 005266 000004          INC      4(SP)
10 004644 005366 000006          DEC      6(SP)
11 004650 003366          BGT      1$
12 004652 000207          2$:  RTS      PC
                                     .SBTTL  NEWFUN
1                                     ;
2                                     ; add an entry to the function table
3                                     ;
4                                     ;
5 004654 062767 000006 000000G NEWFUN: ADD    #6,CURFUN
6 004662 026767 000000G 000000G      CMP      CURFUN,EFUN      ; test for function table overflow
7 004670 101405          BLOS     1$
8 004672 012746 000021          MOV      #TMFUNERR,-(SP) ; report Too Many FUNctions ERROR
9 004676 004737 002352'          JSR      PC,@#ESET
10 004702 005726          TST      (SP)+
11 004704 016777 000000G 000000G 1$:  MOV      NXTVAR,@CURFUN ; mark first variable in new function
12 004712 016700 000000G          MOV      CURFUN,R0
13 004716 016701 000000G          MOV      NXTVAR,R1
14 004722 062701 177762          ADD      #-VLEN-6.,R1
15 004726 010160 000002          MOV      R1,2(R0) ; initialize last variable in new function
16 004732 016700 000000G          MOV      CURFUN,R0 ; return pointer to new function entry
17 004736 016760 000000G 000004          MOV      PRUSED,4(R0) ; insert back pointer
18 004744 000207          RTS      PC
                                     .SBTTL  NEWVAR
1                                     ;
2                                     ; add an entry to the variable table
3                                     ;
4                                     ;
5 004746 004567 001362          NEWVAR: JSR      R5,SAVE
6 004752 016716 000000G          MOV      NXTVAR,(SP)
7 004756 004737 000652'          JSR      PC,@#CANON ; canonicalize the name
8 004762 016700 000000G          MOV      NXTVAR,R0
9 004766 116560 000004 000010          MOVB    4(R5),VLEN(R0) ; stow away its properties
10 004774 116560 000006 000011          MOVB    6(R5),VLEN+1.(R0)
11 005002 016560 000010 000012          MOV      10(R5),VLEN+2.(R0)
12 005010 005765 000012          TST      12(R5)
13 005014 001403          BEQ      1$
14 005016 105765 000004          TSTB     4(R5)
15 005022 001040          BNE      4$
16 005024 016700 000000G          1$:  MOV      NXTVAR,R0 ; if not passed or class 0,
17 005030 016701 000000G          MOV      PRUSED,R1 ; allocate space for it
18 005034 005201          INC      R1

```


tiny-c/11 Version 11-01-01
TCSUBS

19	005036	010160	000014		MOV	R1,VLEN+4.(R0)	
20	005042	116501	000006		MOVB	6(R5),R1	
21	005046	070165	000010		MUL	10(R5),R1	
22	005052	060167	000000G		ADD	R1,PRUSED	
23	005056	026767	000000G	000000G	CMP	PRUSED,EPR	; test for program space overflow
24	005064	101405			BLOS	2\$	
25	005066	012716	000023		MOV	#TMVALERR,(SP)	
26	005072	004737	002352'		JSR	PC,@#ESET	; report Too Many VALues ERRor
27	005076	000412			BR	4\$	
28	005100	016700	000000G	2\$:	MOV	NXTVAR,R0	
29	005104	016002	000014		MOV	VLEN+4.(R0),R2	
30	005110	026702	000000G	3\$:	CMP	PRUSED,R2	
31	005114	002417			BLT	5\$	
32	005116	105012			CLRB	@R2	; clear out new space
33	005120	005202			INC	R2	
34	005122	000772			BR	3\$	
35	005124	005765	000012	4\$:	TST	12(R5)	
36	005130	001411			BEQ	5\$	
37	005132	105765	000004		TSTB	4(R5)	
38	005136	003406			BLE	5\$	
39	005140	016700	000000G		MOV	NXTVAR,R0	; if passed & class>0,
40	005144	016560	000012	000014	MOV	12(R5),VLEN+4.(R0)	; tuck in value
41	005152	000423			BR	6\$	
42	005154	005765	000012	5\$:	TST	12(R5)	
43	005160	001420			BEQ	6\$	
44	005162	105765	000004		TSTB	4(R5)	
45	005166	001015			BNE	6\$	
46	005170	116500	000006		MOVB	6(R5),R0	; if passed & class=0,
47	005174	010016			MOV	R0,(SP)	; MOV iN value
48	005176	010546			MOV	R5,-(SP)	
49	005200	062716	000012		ADD	#12,(SP)	
50	005204	016700	000000G		MOV	NXTVAR,R0	
51	005210	016046	000014		MOV	VLEN+4.(R0),-(SP)	
52	005214	004737	004620'		JSR	PC,@#MOVN	
53	005220	022626			CMP	(SP)+,(SP)+	
54	005222	016700	000000G	6\$:	MOV	CURFUN,R0	
55	005226	016760	000000G	000002	MOV	NXTVAR,2(R0)	; update CURFUN's last variable pointer
56	005234	026767	000000G	000000G	CMP	NXTVAR,EVAR	; test for variable table overflow
57	005242	103404			BLO	7\$	
58	005244	012716	000022		MOV	#TMVARERR,(SP)	
59	005250	004737	002352'		JSR	PC,@#ESET	; report Too Many VARiables ERRor
60	005254	016700	000000G	7\$:	MOV	NXTVAR,R0	
61	005260	016000	000014		MOV	VLEN+4.(R0),R0	
62	005264	062767	000016	000000G	ADD	#VLEN+6.,NXTVAR	; bump NTXVAR pointer
63	005272	000167	001020		JMP	RETURN	

tiny-c/11 Version 11-01-01
TCSUBS

```

1
2
3
4
5
6 005276 122766 000060 000002 NUM:  CMPB    #'0,2(SP)
7 005304 003011                BGT      1$
8 005306 122766 000071 000002    CMPB    #'9,2(SP)
9 005314 002405                BLT      1$
10 005316 116600 000002        MOVB    2(SP),R0
11 005322 162700 000060        SUB      #'0,R0
12 005326 000207                RTS      PC
13 005330 012700 177777    1$:    MOV      #-1,R0
14 005334 000207                RTS      PC
                                .SBTTL   POP
1
2
3
4
5 005336 016700 000000G      POP:    MOV      TOP,R0
6 005342 016000 000004        MOV      4(R0),R0
7 005346 162767 000006 000000G    SUB      #6,TOP
8 005354 000207                RTS      PC
                                .SBTTL   PUSH
1
2
3
4
5 005356 062767 000006 000000G PUSH:  ADD      #6,TOP
6 005364 026767 000000G 000000G    CMP      TOP,ESTACK      ; check for stack overflow
7 005372 103017                BHIS    1$
8 005374 116677 000002 000000G    MOVB    2(SP),@TOP
9 005402 016700 000000G        MOV      TOP,R0
10 005406 116660 000004 000001    MOVB    4(SP),1(R0)
11 005414 116660 000006 000002    MOVB    6(SP),2(R0)
12 005422 016660 000010 000004    MOV      10(SP),4(R0)
13 005430 000405                BR      2$
14 005432 012746 000020    1$:    MOV      #PUSHERR,-(SP)
15 005436 004737 002352'        JSR      PC,@#ESET      ; report a PUSH ERROR
16 005442 005726                TST      (SP)+
17 005444 000207    2$:    RTS      PC

```


tiny-c/11 Version 11-01-01
TCSUBS

```

1
2
3
4
5 005446 004567 000662
6 005452 004767 175132
7 005456 012716 000000G
8 005462 004737 004504'
9 005466 005700
10 005470 001426
11 005472 004767 175112
12 005476 004767 003044
13 005502 010046
14 005504 004767 003036
15 005510 022600
16 005512 002002
17 005514 005016
18 005516 000402
19 005520 012716 000001
20 005524 012746 000002
21 005530 012746 000101
22 005534 005046
23 005536 004737 005356'
24 005542 000167 000550
25 005546 012716 000000G
26 005552 004737 004504'
27 005556 005700
28 005560 001426
29 005562 004767 175022
30 005566 004767 002754
31 005572 010046
32 005574 004767 002746
33 005600 022600
34 005602 003402
35 005604 005016
36 005606 000402
37 005610 012716 000001
38 005614 012746 000002
39 005620 012746 000101
40 005624 005046
41 005626 004737 005356'
42 005632 000167 000460
43 005636 012716 000000G
44 005642 004737 004504'
45 005646 005700
46 005650 001426
47 005652 004767 174732

```

```

.SBTTL RELN
;
; match an expression or logical comparison of expressions
;
RELN: JSR R5,SAVE
      JSR PC,EXPR ; evaluate an expression
      MOV #LESEQL,(SP)
      JSR PC,@#LIT ; is it <= another expression?
      TST R0
      BEQ 3$
      JSR PC,EXPR
      JSR PC,TOPTOI
      MOV R0,-(SP)
      JSR PC,TOPTOI
      CMP (SP)+,R0
      BGE 1$
      CLR (SP)
      BR 2$
1$: MOV #1,(SP)
2$: MOV #2,-(SP)
      MOV #'A,-(SP)
      CLR -(SP)
      JSR PC,@#PUSH
      JMP RETURN
3$: MOV #GRTEQL,(SP)
      JSR PC,@#LIT ; is it >= another expression?
      TST R0
      BEQ 6$
      JSR PC,EXPR
      JSR PC,TOPTOI
      MOV R0,-(SP)
      JSR PC,TOPTOI
      CMP (SP)+,R0
      BLE 4$
      CLR (SP)
      BR 5$
4$: MOV #1,(SP)
5$: MOV #2,-(SP)
      MOV #'A,-(SP)
      CLR -(SP)
      JSR PC,@#PUSH
      JMP RETURN
6$: MOV #EQLEQL,(SP)
      JSR PC,@#LIT ; is it == another expression?
      TST R0
      BEQ 9$
      JSR PC,EXPR

```


tiny-c/11 Version 11-01-01
TCSUBS

48	005656	004767	002664		JSR	PC,TOPTOI	
49	005662	010046			MOV	R0,-(SP)	
50	005664	004767	002656		JSR	PC,TOPTOI	
51	005670	022600			CMP	(SP)+,R0	
52	005672	001402			BEQ	7\$	
53	005674	005016			CLR	(SP)	
54	005676	000402			BR	8\$	
55	005700	012716	000001	7\$:	MOV	#1,(SP)	
56	005704	012746	000002	8\$:	MOV	#2,-(SP)	
57	005710	012746	000101		MOV	#'A,-(SP)	
58	005714	005046			CLR	-(SP)	
59	005716	004737	005356'		JSR	PC,@#PUSH	
60	005722	000167	000370		JMP	RETURN	
61	005726	012716	000000G	9\$:	MOV	#NOTEQL,(SP)	
62	005732	004737	004504'		JSR	PC,@#LIT	; is it != another expression?
63	005736	005700			TST	R0	
64	005740	001426		RELN1:	BEQ	12\$	
65	005742	004767	174642		JSR	PC,EXPR	
66	005746	004767	002574		JSR	PC,TOPTOI	
67	005752	010046			MOV	R0,-(SP)	
68	005754	004767	002566		JSR	PC,TOPTOI	
69	005760	022600			CMP	(SP)+,R0	
70	005762	001002			BNE	10\$	
71	005764	005016			CLR	(SP)	
72	005766	000402			BR	11\$	
73	005770	012716	000001	10\$:	MOV	#1,(SP)	
74	005774	012746	000002	11\$:	MOV	#2,-(SP)	
75	006000	012746	000101		MOV	#'A,-(SP)	
76	006004	005046			CLR	-(SP)	
77	006006	004737	005356'		JSR	PC,@#PUSH	
78	006012	000167	000300		JMP	RETURN	
79	006016	012716	000000G	12\$:	MOV	#GRTHN,(SP)	
80	006022	004737	004504'		JSR	PC,@#LIT	; is it > another expression?
81	006026	005700			TST	R0	
82	006030	001426			BEQ	15\$	
83	006032	004767	174552		JSR	PC,EXPR	
84	006036	004767	002504		JSR	PC,TOPTOI	
85	006042	010046			MOV	R0,-(SP)	
86	006044	004767	002476		JSR	PC,TOPTOI	
87	006050	022600			CMP	(SP)+,R0	
88	006052	002402			BLT	13\$	
89	006054	005016			CLR	(SP)	
90	006056	000402			BR	14\$	
91	006060	012716	000001	13\$:	MOV	#1,(SP)	
92	006064	012746	000002	14\$:	MOV	#2,-(SP)	
93	006070	012746	000101		MOV	#'A,-(SP)	
94	006074	005046			CLR	-(SP)	

95	006076	004737	005356'		JSR	PC,@#PUSH	
96	006102	000167	000210		JMP	RETURN	
97	006106	012716	000000G	15\$:	MOV	#LESTN,(SP)	
98	006112	004737	004504'		JSR	PC,@#LIT	; is it < another expression?
99	006116	005700			R0		
100	006120	001424			TST	18\$	
101	006122	004767	174462		BEQ	18\$	
102	006126	004767	002414		JSR	PC,EXPR	
103	006132	010046			JSR	PC,TOPTOI	
104	006134	004767	002406		MOV	R0,-(SP)	
105	006140	022600			JSR	PC,TOPTOI	
106	006142	003002			CMP	(SP)+,R0	
107	006144	005016			BGT	16\$	
108	006146	000402			CLR	(SP)	
109	006150	012716	000001	16\$:	BR	17\$	
110	006154	012746	000002	17\$:	MOV	#1,(SP)	
111	006160	012746	000101		MOV	#2,-(SP)	
112	006164	005046			MOV	#'A,-(SP)	
113	006166	004737	005356'		CLR	-(SP)	
114	006172	000167	000120	18\$:	JSR	PC,@#PUSH	
1					JMP	RETURN	
2					.SBTTL	REM	
3							
4							
5	006176	016701	000000G	REM:	MOV	CURSOR,R1	; save CURSOR for diagnostic
6	006202	122777	000012 000000G	1\$:	CMPB	#12,CURSOR	; move past newlines
7	006210	001003			BNE	2\$	
8	006212	005267	000000G		INC	CURSOR	
9	006216	000771			BR	1\$	
10	006220	016700	000000G	2\$:	MOV	CURSOR,R0	; check for /*
11	006224	122710	000057		CMPB	#'/(R0)	
12	006230	001031			BNE	4\$	
13	006232	122760	000052 000001		CMPB	#'*,1(R0)	
14	006240	001025			BNE	4\$	
15	006242	062767	000002 000000G		ADD	#2,CURSOR	
16	006250	016700	000000G	3\$:	MOV	CURSOR,R0	; if /*, move to end of line
17	006254	005267	000000G		INC	CURSOR	
18	006260	122710	000012		CMPB	#12,(R0)	
19	006264	001746			BEQ	1\$	; check for blank lines or more remarks
20	006266	026767	000000G 000000G		CMP	PROGEND,CURSOR	; test for run away
21	006274	103365			BHIS	3\$	
22	006276	010167	000000G		MOV	R1,CURSOR	; restore CURSOR for diagnostic
23	006302	012746	000002		MOV	#CURSERR,-(SP)	
24	006306	004737	002352'		JSR	PC,@#ESET	; report a CURSOR ERROR
25	006312	005726			TST	(SP)+	
26	006314	000207		4\$:	RTS	PC	

tiny-c/11 Version 11-01-01
TCSUBS

```

1                                     .SBTTL  RETURN
2                                     ;
3                                     ; unstacks registers and return address ... sets R5 C style
4                                     ;
5 006316 010502 RETURN: MOV      R5,R2
6 006320 014204      MOV      -(R2),R4
7 006322 014203      MOV      -(R2),R3
8 006324 014202      MOV      -(R2),R2
9 006326 010506      MOV      R5,SP
10 006330 012605      MOV      (SP)+,R5
11 006332 000207      RTS      PC
1                                     .SBTTL  SAVE
2                                     ;
3                                     ; stacks return address and registers ... sets R5 C style
4                                     ;
5 006334 010500 SAVE:  MOV      R5,R0
6 006336 010605      MOV      SP,R5
7 006340 010446      MOV      R4,-(SP)
8 006342 010346      MOV      R3,-(SP)
9 006344 010246      MOV      R2,-(SP)
10 006346 004710      JSR      PC,(R0)
1                                     .SBTTL  SETARG
2                                     ;
3                                     ; initialize funtion's locals to value of arguments
4                                     ;
5 006350 004567 177760 SETARG: JSR      R5,SAVE
6 006354 016500 000004      MOV      4(R5),R0      ; get argument's type
7 006360 122760 000101 000001 CMPB    #'A',1(R0)
8 006366 001005      BNE      1$
9 006370 016500 000004      MOV      4(R5),R0      ; if Actual, value is stuff
10 006374 016002 000004      MOV      4(R0),R2
11 006400 000416      BR      2$
12 006402 016500 000004 1$:  MOV      4(R5),R0      ; otherwise, value is
13 006406 016003 000004      MOV      4(R0),R3      ; what stuff points to
14 006412 111300      MOVB     @R3,R0
15 006414 010016      MOV      R0,(SP)
16 006416 010300      MOV      R3,R0
17 006420 116000 000001      MOVB     1(R0),R0
18 006424 010046      MOV      R0,-(SP)
19 006426 004737 010220'    JSR      PC,@#TCTOI
20 006432 005726      TST      (SP)+
21 006434 010002      MOV      R0,R2
22 006436 010216 2$:  MOV      R2,(SP)      ; allocate value
23 006440 116500 000006      MOVB     6(R5),R0
24 006444 010046      MOV      R0,-(SP)
25 006446 004737 010700'    JSR      PC,@#VALLOC
26 006452 000167 177640      JMP      RETURN

```



```

1
2
3
4
5 006456 016701 000000G .SBTTL SKIP
6 006462 012703 000001 ;
7 006466 026767 000000G 000000G 1$: ; move cursor past a balanced pair of 2(SP) and 4(SP)
8 006474 101420 ;
9 006476 126677 000002 000000G SKIP: MOV CURSOR,R1 ; save CURSOR for diagnostic
10 006504 001001 MOV #1,R3 ; initialize parity counter
11 006506 005203 1$: CMP PROGEND,CURSOR ; test for run away
12 006510 016700 000000G BLOS 4$
13 006514 005267 000000G CMPB 2(SP),@CURSOR
14 006520 126610 000004 BNE 2$
15 006524 001001 INC R3 ; increase counter for 2(SP)
16 006526 005303 2$: MOV CURSOR,R0
17 006530 005703 INC CURSOR
18 006532 003410 CMPB 4(SP),(R0)
19 006534 000754 BNE 3$
20 006536 010167 000000G DEC R3 ; decrease counter for 4(SP)
21 006542 012746 000002 3$: TST R3
22 006546 004737 002352' BLE 5$ ; quit on 0 parity
23 006552 005726 BR 1$
24 006554 000207 4$: MOV R1,CURSOR
MOV #CURSERR,-(SP) ; restore CURSOR for diagnostic
JSR PC,@#ESET ; report a CURSOR ERROR
TST (SP)+
5$: RTS PC
.SBTTL SKIPST
1
2
3 ;
4 ; move CURSOR past a possibly compound tiny-c statement
5 006556 004567 177552 ;
6 006562 016702 000000G SKIPST: JSR R5,SAVE
7 006566 004767 177404 MOV CURSOR,R2
8 006572 012716 000000G JSR PC,REM ; move past remarks
9 006576 004737 004504' MOV #LBRACE,(SP) ; if a Left BRACE
10 006602 005700 JSR PC,@#LIT
11 006604 001410 TST R0
12 006606 012716 000135 BEQ 1$
13 006612 012746 000133 MOV #'],(SP)
14 006616 004737 006456' MOV #'[-(SP)
15 006622 005726 JSR PC,@#SKIP ; move to its mate
16 006624 000506 TST (SP)+
17 006626 012716 000000G BR 4$
18 006632 004737 004504' 1$: MOV #IF,(SP) ; move past an if statement
19 006636 005700 JSR PC,@#LIT
20 006640 001426 TST R0
21 006642 012716 000000G BEQ 2$
22 006646 004737 004504' MOV #LPAREN,(SP)
23 006652 012716 000051 JSR PC,@#LIT
MOV #'),(SP)

```


tiny-c/11 Version 11-01-01
TCSUBS

24	006656	012746	000050		MOV	#' , -(SP)	
25	006662	004737	006456'		JSR	PC, @#SKIP	; ... its logical expression
26	006666	005726			TST	(SP) +	
27	006670	004767	177662		JSR	PC, SKIPST	; ... its "true" statement
28	006674	012716	000000G		MOV	#ELSE, (SP)	
29	006700	004737	004504'		JSR	PC, @#LIT	
30	006704	005700			TST	R0	
31	006706	001455			BEQ	4\$	
32	006710	004767	177642		JSR	PC, SKIPST	; ... and its optional else statement
33	006714	000452			BR	4\$	
34	006716	012716	000000G	2\$:	MOV	#WHILE, (SP)	; move past a while statement
35	006722	004737	004504'		JSR	PC, @#LIT	
36	006726	005700			TST	R0	
37	006730	001416			BEQ	3\$	
38	006732	012716	000000G		MOV	#LPAREN, (SP)	
39	006736	004737	004504'		JSR	PC, @#LIT	
40	006742	012716	000051		MOV	#', (SP)	
41	006746	012746	000050		MOV	#' , -(SP)	
42	006752	004737	006456'		JSR	PC, @#SKIP	; ... its logical expression
43	006756	005726			TST	(SP) +	
44	006760	004767	177572		JSR	PC, SKIPST	; ... its "true" statement
45	006764	000426			BR	4\$	
46	006766	122777	000073 000000G	3\$:	CMPB	#', , @CURSOR	; statement ended by a ;
47	006774	001422			BEQ	4\$	
48	006776	122777	000012 000000G		CMPB	#12, @CURSOR	; or a newline
49	007004	001416			BEQ	4\$	
50	007006	005267	000000G		INC	CURSOR	
51	007012	026767	000000G 000000G		CMP	PROGEND, CURSOR	; check for run away
52	007020	103336			BHIS	2\$	
53	007022	010267	000000G		MOV	R2, CURSOR	; restore CURSOR for diagnostic
54	007026	012716	000002		MOV	#CURSERR, (SP)	
55	007032	004737	002352'		JSR	PC, @#ESET	; report a CURSOR ERROR
56	007036	000167	177254		JMP	RETURN	
57	007042	012716	000000G	4\$:	MOV	#SEMI, (SP)	
58	007046	004737	004504'		JSR	PC, @#LIT	; match the end
59	007052	004767	177120		JSR	PC, REM	; and any trailing remarks
60	007056	000167	177234	5\$:	JMP	RETURN	


```

1                                     .SBTTL ST
2                                     ;
3                                     ; evaluate a possibly compound tiny-c statement
4                                     ;
5 007062 004567 177246             ST: JSR R5,SAVE
6 007066 004767 177104             JSR PC,REM ; move past leading remark
7 007072 004767 000000G           JSR PC,STBEGN ; call STBEGN for accounting
8 007076 016767 000000G 000000G MOV CURSOR,STCURS ; save CURSOR
9 007104 012716 000000G           MOV #CHAR,(SP) ; match a char statement
10 007110 004737 004504'          JSR PC,@#LIT
11 007114 005700                  TST R0
12 007116 001431                  BEQ 3$
13 007120 005016                  CLR (SP)
14 007122 012746 000103            MOV #'C,-(SP)
15 007126 004737 010700'          JSR PC,@#VALLOC ; and allocate its variable names
16 007132 005726                  TST (SP)+
17 007134 012716 000000G           1$: MOV #COMMA,(SP)
18 007140 004737 004504'          JSR PC,@#LIT
19 007144 005700                  TST R0
20 007146 001407                  BEQ 2$
21 007150 005016                  CLR (SP)
22 007152 012746 000103            MOV #'C,-(SP)
23 007156 004737 010700'          JSR PC,@#VALLOC
24 007162 005726                  TST (SP)+
25 007164 000763                  BR 1$
26 007166 012716 000000G           2$: MOV #SEMI,(SP)
27 007172 004737 004504'          JSR PC,@#LIT
28 007176 000167 000672            JMP ST3
29 007202 012716 000000G           3$: MOV #INT,(SP) ; match an int statement
30 007206 004737 004504'          JSR PC,@#LIT
31 007212 005700                  TST R0
32 007214 001431                  BEQ 6$
33 007216 005016                  CLR (SP)
34 007220 012746 000111            MOV #'I,-(SP)
35 007224 004737 010700'          JSR PC,@#VALLOC ; and allocate its variable names
36 007230 005726                  TST (SP)+
37 007232 012716 000000G           4$: MOV #COMMA,(SP)
38 007236 004737 004504'          JSR PC,@#LIT
39 007242 005700                  TST R0
40 007244 001407                  BEQ 5$
41 007246 005016                  CLR (SP)
42 007250 012746 000111            MOV #'I,-(SP)
43 007254 004737 010700'          JSR PC,@#VALLOC
44 007260 005726                  TST (SP)+
45 007262 000763                  BR 4$
46 007264 012716 000000G           5$: MOV #SEMI,(SP)
47 007270 004737 004504'          JSR PC,@#LIT
48 007274 000167 000574            JMP ST3

```


tiny-c/11 Version 11-01-01
TCSUBS

```

49 007300 012716 000000G
50 007304 004737 004504'
51 007310 005700
52 007312 001426
53 007314 004767 176656
54 007320 005767 000000G
55 007324 001017
56 007326 005767 000000G
57 007332 001014
58 007334 005767 000000G
59 007340 001011
60 007342 012716 000000G
61 007346 004737 004504'
62 007352 005700
63 007354 001003
64 007356 004767 177500
65 007362 000756
66 007364 000167 000504
67 007370 012716 000000G
68 007374 004737 004504'
69 007400 005700
70 007402 001451
71 007404 012716 000000G
72 007410 004737 004504'
73 007414 004767 170762
74 007420 005767 000000G
75 007424 001022
76 007426 012716 000000G
77 007432 004737 004504'
78 007436 004767 001104
79 007442 005700
80 007444 001414
81 007446 004767 177410
82 007452 012716 000000G
83 007456 004737 004504'
84 007462 005700
85 007464 001402
86 007466 004767 177064
87 007472 000167 000376
88 007476 004767 177054
89 007502 012716 000000G
90 007506 004737 004504'
91 007512 005700
92 007514 001402
93 007516 004767 177340
94 007522 000167 000346
95 007526 012716 000000G
96 007532 004737 004504'

```

```

6$:  MOV    #LBRACE, (SP)      ; evaluate a compound statement
     JSR    PC, @#LIT
     TST    R0
     BEQ    9$
     JSR    PC, REM
7$:  TST    ERR
     BNE    8$
     TST    LEAVE
     BNE    8$
     TST    BRAKE
     BNE    8$
     MOV    #RBRACE, (SP)
     JSR    PC, @#LIT
     TST    R0
     BNE    8$
     JSR    PC, ST              ; by calling ST on the "inside"
     BR     7$
8$:  JMP    ST3
9$:  MOV    #IF, (SP)          ; evaluate an if statement
     JSR    PC, @#LIT
     TST    R0
ST1: BEQ    13$
     MOV    #LPAREN, (SP)
     JSR    PC, @#LIT
     JSR    PC, ASGN            ; ... first its logical condition
     TST    ERR
     BNE    10$
     MOV    #RPAREN, (SP)
     JSR    PC, @#LIT
     JSR    PC, TOPTOI
     TST    R0
     BEQ    11$
     JSR    PC, ST              ; then its "true" statement if true
     MOV    #ELSE, (SP)        ; and skip the optional else
     JSR    PC, @#LIT
     TST    R0
     BEQ    10$
     JSR    PC, SKIPST
10$: JMP    ST3
11$: JSR    PC, SKIPST          ; or skip the "true" statement if false
     MOV    #ELSE, (SP)        ; and evaluate the optional else statement
     JSR    PC, @#LIT
     TST    R0
     BEQ    12$
     JSR    PC, ST
12$: JMP    ST3
13$: MOV    #WHILE, (SP)       ; evaluate a while statement
     JSR    PC, @#LIT

```


tiny-c/11 Version 11-01-01
TCSUBS

97	007536	005700	TST	R0	
98	007540	001450	BEQ	17\$	
99	007542	012716	MOV	#LPAREN, (SP)	
100	007546	004737	JSR	PC, @#LIT	
101	007552	004767	JSR	PC, ASGN	; ... first its logical condition
102	007556	005767	TST	ERR	
103	007562	001027	BNE	14\$	
104	007564	012716	MOV	#RPAREN, (SP)	
105	007570	004737	JSR	PC, @#LIT	
106	007574	004767	JSR	PC, TOPTOI	
107	007600	005700	TST	R0	
108	007602	001424	BEQ	16\$	
109	007604	016702	MOV	STCURS, R2	; remember CURSOR if true
110	007610	016703	MOV	CURSOR, R3	
111	007614	004767	JSR	PC, ST	; and evaluate the following statement
112	007620	005767	TST	BRAKE	; test for declared end-of-while
113	007624	001410	BEQ	15\$	
114	007626	010367	MOV	R3, CURSOR	
115	007632	004767	JSR	PC, SKIPST	; and skip while statement if set
116	007636	005067	CLR	BRAKE	
117	007642	000167	14\$: JMP	ST3	
118	007646	010267	15\$: MOV	R2, CURSOR	
119	007652	000510	BR	ST3	
120	007654	004767	16\$: JSR	PC, SKIPST	; skip statement if while condition is false
121	007660	000505	BR	ST3	
122	007662	012716	17\$: MOV	#SEMI, (SP)	; evaluate the null statement
123	007666	004737	JSR	PC, @#LIT	
124	007672	005700	TST	R0	
125	007674	001401	ST2: BEQ	18\$	
126	007676	000476	BR	ST3	
127	007700	012716	18\$: MOV	#RETERN, (SP)	; return from a function call
128	007704	004737	JSR	PC, @#LIT	
129	007710	005700	TST	R0	
130	007712	001435	BEQ	22\$	
131	007714	012716	MOV	#SEMI, (SP)	; with 0 if ; or newline
132	007720	004737	JSR	PC, @#LIT	
133	007724	005700	TST	R0	
134	007726	001413	BEQ	20\$	
135	007730	005016	19\$: CLR	(SP)	
136	007732	012746	MOV	#2, -(SP)	
137	007736	012746	MOV	#'A, -(SP)	
138	007742	005046	CLR	-(SP)	
139	007744	004737	JSR	PC, @#PUSH	
140	007750	062706	ADD	#6, SP	
141	007754	000410	BR	21\$	
142	007756	012716	20\$: MOV	#LF, (SP)	
143	007762	004737	JSR	PC, @#LIT	
144	007766	005700	TST	R0	

tiny-c/11 Version 11-01-01
TCSUBS

```

145 007770 001357                                BNE      19$
146 007772 004767 170404                          JSR      PC,ASGN      ; or a value if return (expression)
147 007776 012767 000001 000000G 21$:            MOV      #1,LEAVE
148 010004 000433                                BR       ST3
149 010006 012716 000000G 22$:                    MOV      #BREAK,(SP) ; if a break statement
150 010012 004737 004504'                          JSR      PC,@#LIT
151 010016 005700                                TST      R0
152 010020 001404                                BEQ      23$
153 010022 012767 000001 000000G                  MOV      #1,BRAKE      ; set the brake flag
154 010030 000421                                BR       ST3
155 010032 004767 170344 23$:                      JSR      PC,ASGN      ; else match an expression
156 010036 005700                                TST      R0
157 010040 001407                                BEQ      24$
158 010042 004767 000500                          JSR      PC,TOPTOI
159 010046 012716 000000G                          MOV      #SEMI,(SP)
160 010052 004737 004504'                          JSR      PC,@#LIT
161 010056 000406                                BR       ST3
162 010060 012716 000001 24$:                      MOV      #STATERR,(SP) ; if none of the above
163 010064 004737 002352'                          JSR      PC,@#ESET      ; report a STATEment ERROR
164 010070 000167 176222                          JMP      RETURN
165 010074 012716 000000G ST3:                     MOV      #SEMI,(SP)
166 010100 004737 004504'                          JSR      PC,@#LIT
167 010104 004767 176066                          JSR      PC,REM        ; finally, match trailing remarks
168 010110 000167 176202                          JMP      RETURN
1          .SBTTL  SYMNAM
2
3          ;
4          ; tests for symbol name at CURSOR ... if it is, moves CURSOR
5          ; past it and sets FNAME and LNAME to its first and last characters
6 010114 004767 170462 SYMNAM: JSR      PC,BLANKS
7 010120 016767 000000G 000000G                  MOV      CURSOR,FNAME ; set FNAME
8 010126 117700 000000G                          MOV      @CURSOR,R0
9 010132 010046                                MOV      R0,-(SP)
10 010134 004737 000230'                          JSR      PC,@#ALPHA    ; first character must be alphabetic
11 010140 005726                                TST      (SP)+
12 010142 005700                                TST      R0
13 010144 001002                                BNE      1$
14 010146 005000                                CLR      R0          ; return failure
15 010150 000422                                BR       3$
16 010152 005267 000000G 1$:                        INC      CURSOR
17 010156 117700 000000G                          MOV      @CURSOR,R0
18 010162 010046                                MOV      R0,-(SP)
19 010164 004737 000304'                          JSR      PC,@#ALPHANUM ; go until not alphanumeric
20 010170 005726                                TST      (SP)+
21 010172 005700                                TST      R0
22 010174 001401                                BEQ      2$
23 010176 000765                                BR       1$
24 010200 016700 000000G 2$:                        MOV      CURSOR,R0
25 010204 005300                                DEC      R0
26 010206 010067 000000G                          MOV      R0,LNAME      ; set LNAME
27 010212 012700 000001                          MOV      #1,R0        ; return success
28 010216 000207 3$:                                RTS      PC

```


tiny-c/11 Version 11-01-01
TCSUBS

```

1
2
3
4
5 010220 116666 000004 177776
6 010226 116666 000002 177777
7 010234 016600 177776
8 010240 000207
1
2
3
4
5 010242 004567 176066
6 010246 005746
7 010250 004767 172604
8 010254 005767 000000G
9 010260 001130
10 010262 012716 000000G
11 010266 004737 004504'
12 010272 005700
13 010274 001424
14 010276 004767 172556
15 010302 004767 000240
16 010306 010046
17 010310 004767 000232
18 010314 010001
19 010316 070126
20 010320 010116
21 010322 012746 000002
22 010326 012746 000101
23 010332 005046
24 010334 004737 005356'
25 010340 062706 000006
26 010344 000743
27 010346 004767 175624
28 010352 012716 000000G
29 010356 004737 004504'
30 010362 005700
31 010364 001430
32 010366 004767 172466
33 010372 004767 000150
34 010376 010065 177770
35 010402 004767 000140
36 010406 010001
37 010410 005701
38 010412 006700
39 010414 071065 177770

```

```

.SBTTL TCTOI
;
; moves 2(SP) and 4(SP) to low and high of R0
;
TCTOI:  MOV 4(SP),-2(SP)
        MOV 2(SP),-1(SP)
        MOV -2(SP),R0
        RTS PC
.SBTTL TERM
;
; a term is a factor or product of factors
;
TERM:   JSR R5,SAVE
        TST -(SP)
        JSR PC,FACTOR ; evaluate the first factor
1$:     TST ERR
        BNE 4$
        MOV #TIMES,(SP) ; is it factor*factor?
        JSR PC,@#LIT
        TST R0
        BEQ 2$
        JSR PC,FACTOR
        JSR PC,TOPTOI
        MOV R0,-(SP)
        JSR PC,TOPTOI
        MOV R0,R1
        MUL (SP)+,R1
        MOV R1,(SP)
        MOV #2,-(SP)
        MOV #'A,-(SP)
        CLR -(SP)
        JSR PC,@#PUSH
        ADD #6,SP
        BR 1$
2$:     JSR PC,REM
        MOV #DIVIDE,(SP) ; is it factor/factor?
        JSR PC,@#LIT
        TST R0
        BEQ 3$
        JSR PC,FACTOR
        JSR PC,TOPTOI
        MOV R0,-10(R5)
        JSR PC,TOPTOI
        MOV R0,R1
        TST R1
        SXT R0
        DIV -10(R5),R0

```


tiny-c/11 Version 11-01-01
TCSUBS

40	010420	010016		MOV	R0, (SP)	
41	010422	012746	000002	MOV	#2, -(SP)	
42	010426	012746	000101	MOV	#'A, -(SP)	
43	010432	005046		CLR	-(SP)	
44	010434	004737	005356'	JSR	PC, @#PUSH	
45	010440	062706	000006	ADD	#6, SP	
46	010444	000703		BR	1\$	
47	010446	012716	000000G	3\$: MOV	#MODULO, (SP)	; is it factor%factor?
48	010452	004737	004504'	JSR	PC, @#LIT	
49	010456	005700		TST	R0	
50	010460	001430		BEQ	4\$	
51	010462	004767	172372	JSR	PC, FACTOR	
52	010466	004767	000054	JSR	PC, TOPTOI	
53	010472	010065	177770	MOV	R0, -10(R5)	
54	010476	004767	000044	JSR	PC, TOPTOI	
55	010502	010001		MOV	R0, R1	
56	010504	005701		TST	R1	
57	010506	006700		SXT	R0	
58	010510	071065	177770	DIV	-10(R5), R0	
59	010514	010116		MOV	R1, (SP)	
60	010516	012746	000002	MOV	#2, -(SP)	
61	010522	012746	000101	MOV	#'A, -(SP)	
62	010526	005046		CLR	-(SP)	
63	010530	004737	005356'	JSR	PC, @#PUSH	
64	010534	062706	000006	ADD	#6, SP	
65	010540	000645		BR	1\$	
66	010542	000167	175550	4\$: JMP	RETURN	
1				.SBTTL	TOPTOI	
2						
3						
4						
5						
6						
7						
8	010546	016700	000000G	TOPTOI: MOV	TOP, R0	
9	010552	122760	000101 000001	CMPB	#'A, 1(R0)	
10	010560	001003		BNE	1\$	
11	010562	004767	174550	JSR	PC, POP	; if it's an actual, POP & return
12	010566	000443		BR	4\$	
13	010570	122760	000114 000001	1\$: CMPB	#'L, 1(R0)	
14	010576	001032		BNE	3\$	
15	010600	004767	174532	JSR	PC, POP	; if it's an value, resolve it
16	010604	010001		MOV	R0, R1	
17	010606	016700	000000G	MOV	TOP, R0	; if size=1 & class=0, return byte
18	010612	122760	000001 000010	CMPB	#1, 10(R0)	
19	010620	001007		BNE	2\$	
20	010622	016700	000000G	MOV	TOP, R0	

tiny-c/11 Version 11-01-01
TCSUBS

21	010626	105760	000006		TSTB	6(R0)	
22	010632	001002			BNE	2\$	
23	010634	111100			MOVB	@R1,R0	
24	010636	000417			BR	4\$	
25	010640	111100		2\$:	MOVB	@R1,R0	; otherwise, return word
26	010642	010046			MOV	R0,-(SP)	
27	010644	010100			MOV	R1,R0	
28	010646	116000	000001		MOVB	1(R0),R0	
29	010652	010046			MOV	R0,-(SP)	
30	010654	004737	010220'		JSR	PC,@#TCTOI	
31	010660	022626			CMP	(SP)+,(SP)+	
32	010662	000405			BR	4\$	
33	010664	012746	000017	3\$:	MOV	#TOPERR,-(SP)	
34	010670	004737	002352'		JSR	PC,@#ESET	; if neither A nor L, report TOP ERROR
35	010674	005726			TST	(SP)+	
36	010676	000207		4\$:	RTS	PC	
1					.SBTTL	VALLOC	
2							
3							
4							
5	010700	004567	175430		VALLOC: JSR	R5,SAVE	
6	010704	004767	177204		JSR	PC,SYMNAME	; match the symbol name
7	010710	005700			TST	R0	
8	010712	001464			BEQ	6\$	
9	010714	005004			CLR	R4	
10	010716	012716	000000G		MOV	#LPAREN,(SP)	
11	010722	004737	004504'		JSR	PC,@#LIT	; see if it's dimensioned
12	010726	005700			TST	R0	
13	010730	001431			BEQ	2\$	
14	010732	016702	000000G		MOV	FNAME,R2	; yes ... it is
15	010736	016703	000000G		MOV	LNAME,R3	
16	010742	005204			INC	R4	
17	010744	004767	167432		JSR	PC,ASGN	; evaluate its dimension
18	010750	005700			TST	R0	
19	010752	001003			BNE	1\$	
20	010754	005000			CLR	R0	; trouble in evaluation ... return
21	010756	000167	175334		JMP	RETURN	
22	010762	010267	000000G	1\$:	MOV	R2,FNAME	
23	010766	010367	000000G		MOV	R3,LNAME	
24	010772	012716	000000G		MOV	#RPAREN,(SP)	
25	010776	004737	004504'		JSR	PC,@#LIT	
26	011002	004767	177540		JSR	PC,TOPTOI	; get value of dimension
27	011006	005200			INC	R0	
28	011010	010002			MOV	R0,R2	; it's the length
29	011012	000402			BR	3\$	
30	011014	012702	000001	2\$:	MOV	#1,R2	; otherwise length=1
31	011020	122765	000103 000004	3\$:	CMPB	#'C,4(R5)	

tiny-c/11 Version 11-01-01
TCSUBS

```
32 011026 001003
33 011030 012700 000001
34 011034 000402
35 011036 012700 000002
36 011042 016516 000006
37 011046 010246
38 011050 010046
39 011052 010446
40 011054 004737 004746'
41 011060 000167 175232
42 011064 012716 000003
43 011070 004737 002352'
44 011074 000167 175216
45 000001'
```

```
BNE 4$
MOV #1,R0 ; if type=C, size=1
BR 5$
4$: MOV #2,R0 ; otherwise size=2
5$: MOV 6(R5),(SP)
MOV R2,-(SP)
MOV R0,-(SP)
MOV R4,-(SP)
JSR PC,@#NEWVAR ; allocate the new variable
JMP RETURN
6$: MOV #SYMBOLERR,(SP) ; if no symbol name found,
JSR PC,@#ESET ; report a SYMBOL ERROR
JMP RETURN
.END
```


tiny-c/11 Version 11-01-01
TCVBLS

```
1          .SBTTL  GLOBALS
2          .NLIST
3          .TITLE  tiny-c/11
4
5          ; tiny-c/11 assembly constants
6          ;
7          000036      MAXSTACK=30.
8          000144      MAXVAR=100.
9          000036      MAXFUN=30.
10         047040      MAXPR=20000.
11         000062      MAXDEPTH=50.
12         000010      VLEN=8.
13
14         ; tiny-c/11 error codes
15         ;
16         000001      STATERR=1.
17         000002      CURSERR=2.
18         000003      SYMBOLERR=3.
19         000005      RPARERR=5.
20         000006      RANGERR=6.
21         000007      CLASSERR=7.
22         000011      SYNTAXERR=9.
23         000016      LVALERR=14.
24         000017      TCPERR=15.
25         000020      PUSHERR=16.
26         000021      TMFUNERR=17.
27         000022      TMVARERR=18.
28         000023      TMVALERR=19.
29         000024      LINKERR=20.
30         000025      ARGSERR=21.
31         000026      LBRCERR=22.
32         000030      MCERR=24.
33         000031      LOADERR=25.
34         000032      SYMERR=26.
35
36         ; tiny-c/11 global literals
37         ;
38         .GLOBL  LBRACE,RBRACE,LPAREN,RPAREN,COMMA,SEMI,LF,MCLIT,ENDLIB
39         .GLOBL  INT,CHAR,IF,ELSE,WHILE,BREAK,RETERN
40         .GLOBL  EQUAL,PLUS,MINUS,TIMES,DIVIDE,MODULO
41         .GLOBL  EQLEQL,NOTEQL,GRTEQL,LESEQL,GRTHN,LESTHN
42
43         ; tiny-c/11 global variables
44         ;
45         .GLOBL  ERR,ERRAT,LEAVE,BRAKE
46         .GLOBL  TOP,NXTVAR,CURFUN,CURGLO,FNAME,LNAME
47         .GLOBL  APPLVL,STCURS,CURSOR,PRUSED,PROGEN
```


tiny-c/11 Version 11-01-01
TCVBLS

```
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
58  
59  
60  
61  
62  
63  
64  
65  
66 000000  
67 000001  
68 000002  
69 000003  
70 000004  
71 000005  
72 000006  
73 000007  
74 000006  
75 000007  
76
```

```
; tiny-c/11 subroutines  
; .GLOBL TINYC,ADDRVA,ALPHA,ALPHAN,ASGN,ATOI,BLANKS  
; .GLOBL CANON,CEQ,CEQN,CONST,ENTER,ESET  
; .GLOBL EQ,EXPR,FACTOR,FUNDON,LINK,LIT  
; .GLOBL MC,MOVN,NEWFUN,NEWVAR,NUM,POP,PUSH,RELN,REM  
; .GLOBL RETURN,SAVE,SETARG,SKIP,SKIPST,ST,SYMNAM  
; .GLOBL TCTOI,TERM,TOPTOI,VALLOC  
;  
; tiny-c/11 installation vector entries  
; .GLOBL INCH,OUTCH,FLOAD,FOPEN,FREAD,FWRITE,FCLOSE  
; .GLOBL USERMC,PRBEGN,STBEGN,PRDONE  
; .GLOBL BSTACK,ESTACK,BVAR,EVAR,BFUN,EFUN,BPR,EPR,MSTACK  
;  
; tiny-c/11 register assignments  
; R0=%^O0  
; R1=%^O1  
; R2=%^O2  
; R3=%^O3  
; R4=%^O4  
; R5=%^O5  
; R6=%^O6  
; R7=%^O7  
; SP=%^O6  
; PC=%^O7  
; .LIST
```


tiny-c/11 Version 11-01-01
TCVBLS

```
78
79
80
81
82
83
84      000000'
85 000000 000167 000000G
86 000004 000167 000000G
87 000010 000167 000000G
88 000014 000167 000000G
89 000020 000167 000000G
90 000024 000167 000000G
91 000030 000167 000000G
92 000034 000207
93 000036 000000
94 000040 000207
95 000042 000000
96 000044 000207
97 000046 000000
98 000050 000207
99 000052 000000
100
101 000054 001236'
102 000056 001530'
103 000060 001532'
104 000062 004340'
105 000064 004342'
106 000066 004536'
107 000070 004540'
108 000072 053602'
109 000074 056070'
110
111
112
113
114 000076 000000
115 000100 000000
116 000102 000000
117 000104 000000
118 000106 000000
119 000110 000000
120 000112 000000
121 000114 000000
122 000116 000000
123 000120 000000
124 000122 000000
125 000124 000000
126 000126 000000
127 000130 000000
128 000132 000000
```

```
.TITLE tiny-c/11
;
; tiny-c/11 installation vector
;
.SBTTL TCVBLS
.GLOBL GETCHR, PUTCHR, LOAD, OPEN, GETBL, PUTBL, CLOSE
.CSECT TCVBLS
INCH: JMP GETCHR
OUTCH: JMP PUTCHR
FLOAD: JMP LOAD
FOPEN: JMP OPEN
FREAD: JMP GETBL
FWRITE: JMP PUTBL
FCLOSE: JMP CLOSE
USERMC: RTS PC
.WORD 0
PRBEGN: RTS PC
.WORD 0
STBEGN: RTS PC
.WORD 0
PRDONE: RTS PC
.WORD 0
.WORD 0
BSTACK: .WORD STACKB
ESTACK: .WORD STACKE
BVAR: .WORD VARB
EVAR: .WORD VARE
BFUN: .WORD FUNB
EFUN: .WORD FUNE
BPR: .WORD PRB
EPR: .WORD PRE
MSTACK: .WORD MSTCKB
;
; tiny-c/11 global variables
;
.EVEN
ERR: .WORD 0
ERRAT: .WORD 0
LEAVE: .WORD 0
BRAKE: .WORD 0
TOP: .WORD 0
NXTVAR: .WORD 0
CURFUN: .WORD 0
CURGLO: .WORD 0
FNAME: .WORD 0
LNAME: .WORD 0
APPLVL: .WORD 0
STCURS: .WORD 0
CURSOR: .WORD 0
PRUSED: .WORD 0
PROGEN: .WORD 0
```


tiny-c/11 Version 11-01-01
TCVBLS

```
129                                     ;
130                                     ; tiny-c/11 input/output areas
131                                     ;
132                                     .EVEN
133 000134 000000 FILEN0: .RAD50 / /
134 000136 000000 FILEN1: .RAD50 / /
135 000140 000000 FILEN2: .RAD50 / /
136 000142 000000 FILEN3: .RAD50 / /
137                                     .EVEN
138 000144 016336 076604 056754 TCLOAD: .RAD50 /DX0TCLOADTCF/
139                                     .EVEN
140 000154 CSW: .BLKW 10
141                                     .EVEN
142 000174 BLOCK: .BLKW 21
143                                     .EVEN
144 000236 BUFFER: .BLKW 256.
145                                     ;
146                                     ; tiny-c/11 stacks and tables
147                                     ;
148                                     .EVEN
149 001236 STACKB: .BLKB 6*MAXSTACK+6
150                                     .EVEN
151 001530 000000 STACKE: .WORD 0
152 001532 VARB: .BLKB VLEN+6*MAXVAR+VLEN+6
153                                     .EVEN
154 004340 000000 VARE: .WORD 0
155 004342 FUNB: .BLKB 4*MAXFUN+4
156                                     .EVEN
157 004536 000000 FUNE: .WORD 0
158 004540 PRB: .BLKB MAXPR+2
159                                     .EVEN
160 053602 000000 PRE: .WORD 0
161 053604 MSTCKE: .BLKW 12.*MAXDEPTH+2
162                                     .EVEN
163 056070 000000 MSTCKB: .WORD 0
164 000001'                                     .END
```


tiny-c/11 Version 11-01-01
TCMC

```
1          .SBTTL  GLOBALS
2          .NLIST
3          .TITLE  tiny-c/11
4          ;
5          ; tiny-c/11 assembly constants
6          ;
7          000036      MAXSTACK=30.
8          000144      MAXVAR=100.
9          000036      MAXFUN=30.
10         047040      MAXPR=20000.
11         000062      MAXDEPTH=50.
12         000010      VLEN=8.
13         ;
14         ; tiny-c/11 error codes
15         ;
16         000001      STATERR=1.
17         000002      CURSERR=2.
18         000003      SYMBOLERR=3.
19         000005      RPARERR=5.
20         000006      RANGERR=6.
21         000007      CLASSERR=7.
22         000011      SYNTAXERR=9.
23         000016      LVALERR=14.
24         000017      TOPERR=15.
25         000020      PUSHERR=16.
26         000021      TMFUNERR=17.
27         000022      TMVARERR=18.
28         000023      TMVALERR=19.
29         000024      LINKERR=20.
30         000025      ARGSERR=21.
31         000026      LBRCERR=22.
32         000030      MCERR=24.
33         000031      LOADERR=25.
34         000032      SYMERR=26.
35         ;
36         ; tiny-c/11 global literals
37         ;
38         .GLOBL  LBRACE,RBRACE,LPAREN,RPAREN,COMMA,SEMI,LF,MCLIT,ENDLIB
39         .GLOBL  INT,CHAR,IF,ELSE,WHILE,BREAK,RETERN
40         .GLOBL  EQUAL,PLUS,MINUS,TIMES,DIVIDE,MODULO
41         .GLOBL  EQLEQL,NOTEQL,GRTEQL,LESEQL,GRTTHN,LESTHN
42         ;
43         ; tiny-c/11 global variables
44         ;
45         .GLOBL  ERR,ERRAT,LEAVE,BRAKE
46         .GLOBL  TOP,NXTVAR,CURFUN,CURGLO,FNAME,LNAME
47         .GLOBL  APPLVL,STCURS,CURSOR,PRUSED,PROGEN
```


tiny-c/11 Version 11-01-01
TCMC

```
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76

; tiny-c/11 subroutines
;
.GLOBL TINYC,ADDRVA,ALPHA,ALPHAN,ASGN,ATOI,BLANKS
.GLOBL CANON,CEQ,CEQN,CONST,ENTER,ESET
.GLOBL EQ,EXPR,FACTOR,FUNDON,LINK,LIT
.GLOBL MC,MOVN,NEWFUN,NEWVAR,NUM,POP,PUSH,RELN,REM
.GLOBL RETURN,SAVE,SETARG,SKIP,SKIPST,ST,SYMNAM
.GLOBL TCTOI,TERM,TOPTOI,VALLOC
;
; tiny-c/11 installation vector entries
;
.GLOBL INCH,OUTCH,FLOAD,FOPEN,FREAD,FWRITE,FCLOSE
.GLOBL USERMC,PRBEGN,STBEGN,PRDONE
.GLOBL BSTACK,ESTACK,BVAR,EVAR,BFUN,EFUN,BPR,EPR,MSTACK
;
; tiny-c/11 register assignments
;
R0=%^00
R1=%^01
R2=%^02
R3=%^03
R4=%^04
R5=%^05
R6=%^06
R7=%^07
SP=%^06
PC=%^07
.LIST
```


tiny-c/11 Version 11-01-01
TCMC

```

78                                     .TITLE tiny-c/11
79                                     .SBTTL MC
80                                     .CSECT TCMC
81
82                                     ;
83                                     ; machine call (MC) subroutines for tiny-c/11
84                                     ;
85                                     ; function number=-10(R5), no. of arguments=4(R5)
86                                     ;
86 000000 004567 000000G MC: JSR R5,SAVE
87 000004 005765 000004 TST 4(R5)
88 000010 003002 BGT 1$
89 000012 000167 003122 JMP MC99
90 000016 162706 000036 1$: SUB #36,SP ; make some room for locals
91 000022 004767 000000G JSR PC,POP ; get the function number off the stack
92 000026 010065 177770 MOV R0,-10(R5)
93 000032 005365 000004 DEC 4(R5)
94 000036 022765 001750 177770 CMP #1000.,-10(R5) ; test for user MC
95 000044 101012 BHI MC1
96 000046 016516 000004 MOV 4(R5),(SP)
97 000052 016516 177770 MOV -10(R5),(SP)
98 000056 162746 001750 SUB #1000.,-(SP)
99 000062 004767 000000G JSR PC,USERMC
100 000066 000167 000000G JMP RETURN
101
102                                     ;
103                                     ; MC 1 ... put a character to the standard output
104 000072 022765 000001 177770 MC1: CMP #1,-10(R5)
105 000100 001030 BNE MC2
106 000102 022765 000001 000004 CMP #1,4(R5)
107 000110 001402 BEQ 1$
108 000112 000167 003002 JMP MC98
109 000116 004767 000000G 1$: JSR PC,TOPTOI
110 000122 110065 177766 MOV R0,-12(R5)
111 000126 116516 177766 MOVB -12(R5),(SP)
112 000132 004737 000000G JSR PC,@#OUTCH
113 000136 005016 CLR (SP)
114 000140 012746 000002 MOV #2,-(SP)
115 000144 012746 000101 MOV #101,-(SP)
116 000150 005046 CLR -(SP)
117 000152 004737 000000G JSR PC,@#PUSH
118 000156 000167 000000G JMP RETURN
```


tiny-c/11 Version 11-01-01
TCMC

```
119                                     ;
120                                     ; MC 2 ... get a character from the standard input
121                                     ;
122 000162 022765 000002 177770 MC2:  CMP    #2,-10(R5)
123 000170 001021                BNE    MC3
124 000172 005765 000004                TST    4(R5)
125 000176 001402                BEQ    1$
126 000200 000167 002714                JMP    MC98
127 000204 004767 000000G              1$:  JSR    PC,INCH
128 000210 010016                MOV    R0,(SP)
129 000212 012746 000001                MOV    #1,-(SP)
130 000216 012746 000101                MOV    #101,-(SP)
131 000222 005046                CLR    -(SP)
132 000224 004737 000000G              JSR    PC,@#PUSH
133 000230 000167 000000G              JMP    RETURN
134                                     ;
135                                     ; MC 3 ... file open(r/w, name, size, channel)
136                                     ;
137 000234 022765 000003 177770 MC3:  CMP    #3,-10(R5)
138 000242 001040                BNE    MC4
139 000244 022765 000004 000004        CMP    #4,4(R5)
140 000252 001402                BEQ    1$
141 000254 000167 002640                JMP    MC98
142 000260 004767 000000G              1$:  JSR    PC,TOPTOI
143 000264 010016                MOV    R0,(SP)
144 000266 004767 000000G              JSR    PC,TOPTOI
145 000272 010046                MOV    R0,-(SP)
146 000274 004767 000000G              JSR    PC,TOPTOI
147 000300 010046                MOV    R0,-(SP)
148 000302 004767 000000G              JSR    PC,TOPTOI
149 000306 010046                MOV    R0,-(SP)
150 000310 004737 000000G              JSR    PC,@#FOPEN
151 000314 062706 000006                ADD    #6,SP
152 000320 010016                MOV    R0,(SP)
153 000322 012746 000001                MOV    #1,-(SP)
154 000326 012746 000101                MOV    #101,-(SP)
155 000332 005046                CLR    -(SP)
156 000334 004737 000000G              JSR    PC,@#PUSH
157 000340 000167 000000G              JMP    RETURN
```



```

158
159 ; MC 4 ... file read(to, channel)
160 ;
161 000344 022765 000004 177770 MC4: CMP #4,-10(R5)
162 000352 001031 BNE MC5
163 000354 022765 000002 000004 CMP #2,4(R5)
164 000362 001402 BEQ 1$
165 000364 000167 002530 JMP MC98
166 000370 004767 000000G 1$: JSR PC,TOPTOI
167 000374 010016 MOV R0,(SP)
168 000376 004767 000000G JSR PC,TOPTOI
169 000402 010046 MOV R0,-(SP)
170 000404 004737 000000G JSR PC,@#FREAD
171 000410 005726 TST (SP)+
172 000412 010016 MOV R0,(SP)
173 000414 012746 000001 MOV #1,-(SP)
174 000420 012746 000101 MOV #101,-(SP)
175 000424 005046 CLR -(SP)
176 000426 004737 000000G JSR PC,@#PUSH
177 000432 000167 000000G JMP RETURN
178
179 ; MC 5 ... file write(from, to, channel)
180 ;
181 000436 022765 000005 177770 MC5: CMP #5,-10(R5)
182 000444 001034 BNE MC6
183 000446 022765 000003 000004 CMP #3,4(R5)
184 000454 001402 BEQ 1$
185 000456 000167 002436 JMP MC98
186 000462 004767 000000G 1$: JSR PC,TOPTOI
187 000466 010016 MOV R0,(SP)
188 000470 004767 000000G JSR PC,TOPTOI
189 000474 010046 MOV R0,-(SP)
190 000476 004767 000000G JSR PC,TOPTOI
191 000502 010046 MOV R0,-(SP)
192 000504 004737 000000G JSR PC,@#FWRITE
193 000510 022626 CMP (SP)+,(SP)+
194 000512 010016 MOV R0,(SP)
195 000514 012746 000001 MOV #1,-(SP)
196 000520 012746 000101 MOV #101,-(SP)
197 000524 005046 CLR -(SP)
198 000526 004737 000000G JSR PC,@#PUSH
199 000532 000167 000000G JMP RETURN

```


tiny-c/11 Version 11-01-01
TCMC

```

200
201 ; MC 6 ... file close(channel)
202 ;
203 000536 022765 000006 177770 MC6: CMP #6,-10(R5)
204 000544 001025 BNE MC7
205 000546 022765 000001 000004 CMP #1,4(R5)
206 000554 001402 BEQ 1$
207 000556 000167 002336 JMP MC98
208 000562 004767 000000G 1$: JSR PC,TOPTOI
209 000566 010016 MOV R0,(SP)
210 000570 004767 000000G JSR PC,FCLOSE
211 000574 010016 MOV R0,(SP)
212 000576 012746 000001 MOV #1,-(SP)
213 000602 012746 000101 MOV #101,-(SP)
214 000606 005046 CLR -(SP)
215 000610 004737 000000G JSR PC,@#PUSH
216 000614 000167 000000G JMP RETURN
217 ;
218 ; MC 7 ... copy a block(start, end, bytes to new start)
219 ;
220 000620 022765 000007 177770 MC7: CMP #7,-10(R5)
221 000626 001076 BNE MC8
222 000630 022765 000003 000004 CMP #3,4(R5)
223 000636 001402 BEQ 1$
224 000640 000167 002254 JMP MC98
225 000644 004767 000000G 1$: JSR PC,TOPTOI
226 000650 010065 177742 MOV R0,-36(R5)
227 000654 004767 000000G JSR PC,TOPTOI
228 000660 010065 177746 MOV R0,-32(R5)
229 000664 004767 000000G JSR PC,TOPTOI
230 000670 010065 177750 MOV R0,-30(R5)
231 000674 005765 177742 TST -36(R5)
232 000700 003415 BLE 3$
233 000702 026565 177746 177750 2$: CMP -32(R5),-30(R5)
234 000710 103411 BLO 3$
235 000712 016500 177746 MOV -32(R5),R0
236 000716 066500 177742 ADD -36(R5),R0
237 000722 117510 177746 MOVB @-32(R5),(R0)
238 000726 005365 177746 DEC -32(R5)
239 000732 000763 BR 2$
240 000734 005765 177742 3$: TST -36(R5)
241 000740 002015 BGE 5$
242 000742 026565 177746 177750 4$: CMP -32(R5),-30(R5)
243 000750 103411 BLO 5$
244 000752 016500 177750 MOV -30(R5),R0
245 000756 066500 177742 ADD -36(R5),R0
246 000762 117510 177750 MOVB @-30(R5),(R0)

```


tiny-c/11 Version 11-01-01
TCMC

247	000766	005265	177750		INC	-30(R5)
248	000772	000763			BR	4\$
249	000774	005016		5\$:	CLR	(SP)
250	000776	012746	000002		MOV	#2,-(SP)
251	001002	012746	000101		MOV	#101,-(SP)
252	001006	005046			CLR	-(SP)
253	001010	004737	000000G		JSR	PC,@#PUSH
254	001014	062706	000006		ADD	#6,SP
255	001020	000167	000000G		JMP	RETURN
256						
257						
258						
259	001024	022765	000010	177770	MC8:	CMP #8,-10(R5)
260	001032	001055			BNE	MC9
261	001034	022765	000003	000004	CMP	#3,4(R5)
262	001042	001402			BEQ	1\$
263	001044	000167	002050		JMP	MC98
264	001050	004767	000000G		JSR	PC,TOPTOI
265	001054	110065	177766		MOVB	R0,-12(R5)
266	001060	004767	000000G		JSR	PC,TOPTOI
267	001064	010065	177746		MOV	R0,-32(R5)
268	001070	004767	000000G		JSR	PC,TOPTOI
269	001074	010065	177750		MOV	R0,-30(R5)
270	001100	005065	177742		CLR	-36(R5)
271	001104	026565	177746	177750	2\$:	CMP -32(R5),-30(R5)
272	001112	103412			BLO	3\$
273	001114	016500	177750		MOV	-30(R5),R0
274	001120	005265	177750		INC	-30(R5)
275	001124	126510	177766		CMPB	-12(R5),(R0)
276	001130	001365			BNE	2\$
277	001132	005265	177742		INC	-36(R5)
278	001136	000762			BR	2\$
279	001140	016516	177742		3\$:	MOV -36(R5),(SP)
280	001144	012746	000002		MOV	#2,-(SP)
281	001150	012746	000101		MOV	#101,-(SP)
282	001154	005046			CLR	-(SP)
283	001156	004737	000000G		JSR	PC,@#PUSH
284	001162	000167	000000G		JMP	RETURN

; MC 8 ... count character instances(from, to, character)

; MC8: CMP #8,-10(R5)
BNE MC9
CMP #3,4(R5)
BEQ 1\$
JMP MC98
1\$: JSR PC,TOPTOI
MOVB R0,-12(R5)
JSR PC,TOPTOI
MOV R0,-32(R5)
JSR PC,TOPTOI
MOV R0,-30(R5)
CLR -36(R5)
2\$: CMP -32(R5),-30(R5)
BLO 3\$
MOV -30(R5),R0
INC -30(R5)
CMPB -12(R5),(R0)
BNE 2\$
INC -36(R5)
BR 2\$
3\$: MOV -36(R5),(SP)
MOV #2,-(SP)
MOV #101,-(SP)
CLR -(SP)
JSR PC,@#PUSH
JMP RETURN

tiny-c/11 Version 11-01-01
TCMC

```

285
286 ; MC 9 ... find nth character instance(from, to, character, n)
287 ;
288 001166 022765 000011 177770 MC9: CMP #9.,-10(R5)
289 001174 001114 BNE MC10
290 001176 022765 000004 000004 CMP #4,4(R5)
291 001204 001402 BEQ 1$
292 001206 000167 001706 JMP MC98
293 001212 004767 000000G 1$: JSR PC,TOPTOI
294 001216 010065 177734 MOV R0,-44(R5)
295 001222 012716 000002 MOV #2,(SP)
296 001226 016546 177734 MOV -44(R5),-(SP)
297 001232 010546 MOV R5, -(SP)
298 001234 062716 177744 ADD #-34,(SP)
299 001240 004737 000000G JSR PC,@#MOVN
300 001244 022626 CMP (SP)+,(SP)+
301 001246 004767 000000G JSR PC,TOPTOI
302 001252 110065 177766 MOVB R0,-12(R5)
303 001256 004767 000000G JSR PC,TOPTOI
304 001262 010065 177746 MOV R0,-32(R5)
305 001266 004767 000000G JSR PC,TOPTOI
306 001272 010065 177750 MOV R0,-30(R5)
307 001276 012765 177777 177742 MOV #-1,-36(R5)
308 001304 005765 177744 2$: TST -34(R5)
309 001310 003420 BLE 4$
310 001312 026565 177746 177750 CMP -32(R5),-30(R5)
311 001320 103414 BLO 4$
312 001322 016500 177750 MOV -30(R5),R0
313 001326 005265 177750 INC -30(R5)
314 001332 126510 177766 CMPB -12(R5),(R0)
315 001336 001002 BNE 3$
316 001340 005365 177744 DEC -34(R5)
317 001344 005265 177742 3$: INC -36(R5)
318 001350 000755 BR 2$
319 001352 016516 177742 4$: MOV -36(R5),(SP)
320 001356 012746 000002 MOV #2, -(SP)
321 001362 012746 000101 MOV #101, -(SP)
322 001366 005046 CLR -(SP)
323 001370 004737 000000G JSR PC,@#PUSH
324 001374 062706 000006 ADD #6,SP
325 001400 012716 000002 MOV #2,(SP)
326 001404 010546 MOV R5, -(SP)
327 001406 062716 177744 ADD #-34,(SP)
328 001412 016546 177734 MOV -44(R5), -(SP)
329 001416 004737 000000G JSR PC,@#MOVN
330 001422 000167 000000G JMP RETURN

```



```

331
332 ; MC 10 ... exit to the RT-11 monitor
333 ;
334 001426 022765 000012 177770 MC10: CMP #12,-10(R5)
335 001434 001001 BNE MC11
336 001436 104350 EMT ^O350
337 ;
338 ; MC 11 ... enter an application program
339 ;
340 001440 022765 000013 177770 MC11: CMP #11.,-10(R5)
341 001446 001162 BNE MC12
342 001450 022765 000004 000004 CMP #4,4(R5)
343 001456 001402 BEQ 1$
344 001460 000167 001434 JMP MC98
345 001464 016765 000000G 177764 1$: MOV CURSOR,-14(R5)
346 001472 016765 000000G 177756 MOV PROGEND,-22(R5)
347 001500 016765 000000G 177766 MOV CURGLOBA,-12(R5)
348 001506 004767 000000G JSR PC,TOPTOI
349 001512 010067 000000G MOV R0,CURSOR
350 001516 010065 177762 MOV R0,-16(R5)
351 001522 004767 000000G JSR PC,TOPTOI
352 001526 010067 000000G MOV R0,PROGEND
353 001532 004767 000000G JSR PC,LINK
354 001536 016767 000000G 000000G MOV CURFUN,CURGLOBA
355 001544 004767 000000G JSR PC,TOPTOI
356 001550 010067 000000G MOV R0,CURSOR
357 001554 004767 000000G JSR PC,NEWFUN
358 001560 004767 000000G JSR PC,TOPTOI
359 001564 010065 177760 MOV R0,-20(R5)
360 001570 005767 000000G TST ERR
361 001574 001012 BNE 2$
362 001576 005267 000000G INC APPLVL
363 001602 004767 000000G JSR PC,PRBEGN
364 001606 004767 000000G JSR PC,ST
365 001612 004767 000000G JSR PC,PRDONE
366 001616 005367 000000G DEC APPLVL
367 001622 004767 000000G 2$: JSR PC,FUNDONE
368 001626 004767 000000G JSR PC,FUNDONE
369 001632 016700 000000G MOV CURSOR,R0
370 001636 166500 177762 SUB -16(R5),R0
371 001642 010065 177752 MOV R0,-26(R5)
372 001646 005767 000000G TST ERR
373 001652 001406 BEQ 3$
374 001654 016700 000000G MOV ERRAT,R0
375 001660 166500 177762 SUB -16(R5),R0
376 001664 010065 177752 MOV R0,-26(R5)
377 001670 012716 000002 3$: MOV #2,(SP)
378 001674 010546 MOV R5,-(SP)

```


tiny-c/11 Version 11-01-01
TCMC

379	001676	062716	177752		ADD	#-26, (SP)
380	001702	016546	177760		MOV	-20(R5), -(SP)
381	001706	062716	000002		ADD	#2, (SP)
382	001712	004737	000000G		JSR	PC, @#MOVN
383	001716	022626			CMP	(SP)+, (SP)+
384	001720	012716	000002		MOV	#2, (SP)
385	001724	012746	000000G		MOV	#ERR, -(SP)
386	001730	016546	177760		MOV	-20(R5), -(SP)
387	001734	004737	000000G		JSR	PC, @#MOVN
388	001740	022626			CMP	(SP)+, (SP)+
389	001742	016567	177764	000000G	MOV	-14(R5), CURSOR
390	001750	016567	177756	000000G	MOV	-22(R5), PROGEND
391	001756	016567	177766	000000G	MOV	-12(R5), CURGLOBA
392	001764	005067	000000G		CLR	ERR
393	001770	005016			CLR	(SP)
394	001772	012746	000002		MOV	#2, -(SP)
395	001776	012746	000101		MOV	#101, -(SP)
396	002002	005046			CLR	-(SP)
397	002004	004737	000000G		JSR	PC, @#PUSH
398	002010	000167	000000G		JMP	RETURN
399						
400						
401						
402	002014	022765	000014	177770	MC12:	CMP #12., -10(R5)
403	002022	001012			BNE	MC13
404	002024	005016			CLR	(SP)
405	002026	012746	000002		MOV	#2, -(SP)
406	002032	012746	000101		MOV	#101, -(SP)
407	002036	005046			CLR	-(SP)
408	002040	004737	000000G		JSR	PC, @#PUSH
409	002044	000167	000000G		JMP	RETURN
410						
411						
412						
413	002050	022765	000015	177770	MC13:	CMP #13., -10(R5)
414	002056	001042			BNE	MC14
415	002060	022765	000002	000004	CMP	#2, 4(R5)
416	002066	001402			BEQ	1\$
417	002070	000167	001024		JMP	MC98
418	002074	004767	000000G		JSR	PC, TOPTOI
419	002100	010065	177766		MOV	R0, -12(R5)
420	002104	004767	000000G		JSR	PC, TOPTOI
421	002110	010001			MOV	R0, R1
422	002112	020165	177766		CMP	R1, -12(R5)
423	002116	101010			BHI	4\$
424	002120	111116			MOVB	@R1, (SP)
425	002122	001002			BNE	3\$

Address	Hex	Dec	Label	Instruction	Comment
426	002124	112716		MOV	#1, (SP)
427	002130	004737	000000G	JSR	PC, @#OUTCH
428	002134	005201		INC	R1
429	002136	000765		BR	2\$
430	002140	005016		4\$: CLR	(SP)
431	002142	012746	000002	MOV	#2, -(SP)
432	002146	012746	000101	MOV	#101, -(SP)
433	002152	005046		CLR	-(SP)
434	002154	004737	000000G	JSR	PC, @#PUSH
435	002160	000167	000000G	JMP	RETURN
436					
437					
438					
439	002164	022765	000016 177770	MC14: CMP	#14., -10 (R5)
440	002172	001402		BEQ	1\$
441	002174	000167	000720	JMP	MC98
442	002200	022765	000001 000004	1\$: CMP	#1, 4 (R5)
443	002206	001402		BEQ	2\$
444	002210	000167	000704	JMP	MC98
445	002214	004767	000000G	2\$: JSR	PC, TOPTOI
446	002220	010016		MOV	R0, (SP)
447	002222	012746	000000	MOV	#0, -(SP)
448	002226	005766	000002	TST	2 (SP)
449	002232	100014		BPL	4\$
450	002234	005466	000002	NEG	2 (SP)
451	002240	102004		BVC	3\$
452	002242	012716	044000	MOV	#44000, (SP)
453	002246	005066	000002	CLR	2 (SP)
454	002252	012746	000055	3\$: MOV	#1, -(SP)
455	002256	004737	000000G	JSR	PC, @#OUTCH
456	002262	005726		TST	(SP) +
457	002264	016604	000002	4\$: MOV	2 (SP), R4
458	002270	005002		CLR	R2
459	002272	011603		MOV	(SP), R3
460	002274	005746		TST	-(SP)
461	002276	012700	000037	MOV	#37, R0
462	002302	005704		TST	R4
463	002304	001006		BNE	5\$
464	002306	012716	000060	MOV	#10, (SP)
465	002312	004737	000000G	JSR	PC, @#OUTCH
466	002316	000167	000460	JMP	EX14
467	002322	005300		5\$: DEC	R0
468	002324	006304		ASL	R4
469	002326	006103		ROL	R3
470	002330	030327	040000	6\$: BIT	R3, #40000
471	002334	001772		BEQ	5\$
472	002336	005700		TST	R0
473	002340	003016		BGT	7\$

tiny-c/11 Version 11-01-01
TCMC

474 002342 006203
475 002344 006004
476 002346 006203
477 002350 006004
478 002352 006203
479 002354 006004
480 002356 006203
481 002360 006004
482 002362 062700 000004
483 002366 004767 000434
484 002372 005302
485 002374 000755
486 002376 020027 000004
487 002402 003407
488 002404 004767 000446
489 002410 005202
490 002412 000746
491 002414 005200
492 002416 006203
493 002420 006004
494 002422 020027 000004
495 002426 002772
496 002430 020327 050000
497 002434 103363
498 002436 062704 001250
499 002442 103010
500 002444 005203
501 002446 020327 050000
502 002452 103404
503 002454 012703 040000
504 002460 005004
505 002462 005202
506 002464 012700 000006
507 002470 010346
508 002472 000316
509 002474 006016
510 002476 006016
511 002500 006016
512 002502 042716 177760
513 002506 062716 000060
514 002512 042703 174000
515 002516 004767 000304
516 002522 005300
517 002524 003361
518 002526 010603
519 002530 062703 000014
520 002534 020227 177776
521 002540 002404

ASR R3
ROR R4
ASR R3
ROR R4
ASR R3
ROR R4
ASR R3
ROR R4
ADD #4,R0
JSR PC,MUL
DEC R2
BR 6\$
7\$: CMP R0,#4
BLE 10\$
8\$: JSR PC,DIV
INC R2
BR 6\$
9\$: INC R0
ASR R3
ROR R4
10\$: CMP R0,#4
BLT 9\$
CMP R3,#50000
BHS 8\$
ADD #1250,R4
RESET1: BCC 11\$
INC R3
CMP R3,#50000
BLO 11\$
MOV #40000,R3
CLR R4
INC R2
11\$: MOV #6,R0
12\$: MOV R3,-(SP)
SWAB (SP)
ROR (SP)
ROR (SP)
ROR (SP)
BIC #177760,(SP)
ADD #'0,(SP)
BIC #174000,R3
JSR PC,MUL
DEC R0
BGT 12\$
MOV SP,R3
ADD #14,R3
CMP R2,#-2
BLT 13\$

522	002542	001456		BEQ	18\$
523	002544	020227	000006	CMP	R2, #6
524	002550	002463		BLT	19\$
525	002552	014346		13\$: MOV	-(R3), -(SP)
526	002554	004737	000000G	JSR	PC, @#OUTCH
527	002560	012716	000056	MOV	#'. , (SP)
528	002564	004737	000000G	JSR	PC, @#OUTCH
529	002570	012704	000005	MOV	#5, R4
530	002574	014316		14\$: MOV	-(R3), (SP)
531	002576	004737	000000G	JSR	PC, @#OUTCH
532	002602	005304		DEC	R4
533	002604	003373		BGT	14\$
534	002606	012716	000105	MOV	#'E, (SP)
535	002612	004737	000000G	JSR	PC, @#OUTCH
536	002616	012716	000053	MOV	#'+, (SP)
537	002622	005702		TST	R2
538	002624	100003		BPL	15\$
539	002626	012716	000055	MOV	#'-, (SP)
540	002632	005402		NEG	R2
541	002634	004737	000000G	15\$: JSR	PC, @#OUTCH
542	002640	012716	000060	MOV	#'0, (SP)
543	002644	162702	000012	16\$: SUB	#12, R2
544	002650	100402		BMI	17\$
545	002652	005200		INC	R0
546	002654	000773		BR	16\$
547	002656	004737	000000G	17\$: JSR	PC, @#OUTCH
548	002662	010216		MOV	R2, (SP)
549	002664	062716	000072	ADD	#'0+12, (SP)
550	002670	004737	000000G	JSR	PC, @#OUTCH
551	002674	000167	000102	JMP	EX14
552	002700	012716	000056	18\$: MOV	#'. , (SP)
553	002704	004737	000000G	JSR	PC, @#OUTCH
554	002710	012716	000060	MOV	#'0, (SP)
555	002714	004737	000000G	JSR	PC, @#OUTCH
556	002720	012704	000006	19\$: MOV	#6, R4
557	002724	010600		MOV	SP, R0
558	002726	022027	000060	20\$: CMP	(R0)+, #'0
559	002732	001002		BNE	21\$
560	002734	005304		DEC	R4
561	002736	000773		BR	20\$
562	002740	020402		21\$: CMP	R4, R2
563	002742	003002		BGT	22\$
564	002744	010204		MOV	R2, R4
565	002746	005204		INC	R4
566	002750	005202		22\$: INC	R2
567	002752	005202		INC	R2
568	002754	005302		23\$: DEC	R2
569	002756	001004		BNE	24\$

tiny-c/11 Version 11-01-01
TCMC

```
570 002760 012716 000056
571 002764 004737 000000G
572 002770 014316
573 002772 004737 000000G
574 002776 005304
575 003000 003365
576 003002 005016
577 003004 012746 000002
578 003010 012746 000101
579 003014 005046
580 003016 004737 000000G
581 003022 000167 000000G
582 003026 010346
583 003030 010446
584 003032 006304
585 003034 006103
586 003036 006304
587 003040 006103
588 003042 062604
589 003044 005503
590 003046 062603
591 003050 006304
592 003052 006103
593 003054 000207
594 003056 012746 000034
595 003062 022703 050000
596 003066 101002
597 003070 062703 130000
598 003074 006104
599 003076 006103
600 003100 005316
601 003102 003367
602 003104 042703 170000
603 003110 005726
604 003112 000207
605 003114 000167 000000G
```

```
24$: MOV #', (SP)
      JSR PC,@#OUTCH
      MOV -(R3), (SP)
      JSR PC,@#OUTCH
      DEC R4
      BGT 23$
EX14: CLR (SP)
      MOV #2, -(SP)
      MOV #101, -(SP)
      CLR -(SP)
      JSR PC,@#PUSH
      JMP RETURN
MUL:  MOV R3, -(SP)
      MOV R4, -(SP)
      ASL R4
      ROL R3
      ASL R4
      ROL R3
      ADD (SP)+, R4
      ADC R3
      ADD (SP)+, R3
      ASL R4
      ROL R3
      RTS PC
DIV:  MOV #34, -(SP)
25$: CMP #50000, R3
      BHI 26$
      ADD #130000, R3
26$: ROL R4
      ROL R3
      DEC (SP)
      BGT 25$
      BIC #170000, R3
      TST (SP)+
      RTS PC
      JMP RETURN
```


tiny-c/11 Version 11-01-01
TCMC

```
606
607
608
609 003120 005765 000004
610 003124 003405
611 003126 004767 000000G
612 003132 005365 000004
613 003136 000770
614
615
616
617 003140 012716 000030
618 003144 004737 000000G
619 003150 005016
620 003152 012746 000002
621 003156 012746 000101
622 003162 005046
623 003164 004737 000000G
624 003170 000167 000000G
625 000001
```

```
;
; wrong number of arguments ... pop 'em off the stack
;
```

```
MC98: TST      4(R5)
      BLE      MC99
      JSR      PC,POP
      DEC      4(R5)
      BR       MC98
```

```
;
; MC error
;
```

```
MC99: MOV      #MCERR, (SP)
      JSR      PC,@#ESET
      CLR      (SP)
      MOV      #2,-(SP)
      MOV      #101,-(SP)
      CLR      -(SP)
      JSR      PC,@#PUSH
      JMP      RETURN
      .END
```

TCMC
tiny-c/11 Version 11-01-01

tiny-c/11 Version 11-01-01
TCIO

```
1          .SBTTL  GLOBALS
2          .NLIST
3          .TITLE  tiny-c/11
4          ;
5          ; tiny-c/11 assembly constants
6          ;
7          000036          MAXSTACK=30.
8          000144          MAXVAR=100.
9          000036          MAXFUN=30.
10         047040          MAXPR=20000.
11         000062          MAXDEPTH=50.
12         000010          VLEN=8.
13
14          ;
15          ; tiny-c/11 error codes
16          ;
17         000001          STATERR=1.
18         000002          CURSERR=2.
19         000003          SYMBOLERR=3.
20         000005          RPARERR=5.
21         000006          RANGERR=6.
22         000007          CLASSERR=7.
23         000011          SYNTAXERR=9.
24         000016          LVALERR=14.
25         000017          TOPERR=15.
26         000020          PUSHERR=16.
27         000021          TMFUNERR=17.
28         000022          TMVARERR=18.
29         000023          TMVALERR=19.
30         000024          LINKERR=20.
31         000025          ARGSERR=21.
32         000026          LBRRCERR=22.
33         000030          MCERR=24.
34         000031          LOADERR=25.
35         000032          SYMERRA=26.
36
37          ;
38          ; tiny-c/11 global literals
39          ;
40         .GLOBL  LBRACE,RBRACE,LPAREN,RPAREN,COMMA,SEMI,LF,MCLIT,ENDLIB
41         .GLOBL  INT,CHAR,IF,ELSE,WHILE,BREAK,RETERN
42         .GLOBL  EQUAL,PLUS,MINUS,TIMES,DIVIDE,MODULO
43         .GLOBL  EQLEQL,NOTEQL,GRTEQL,LESEQL,GRTHN,LESTHN
44
45          ;
46          ; tiny-c/11 global variables
47          ;
48         .GLOBL  ERR,ERRAT,LEAVE,BRAKE
49         .GLOBL  TOP,NXTVAR,CURFUN,CURGLO,FNAME,LNAME
50         .GLOBL  APPLVL,STCURS,CURSOR,PRUSED,PROGEN
```


tiny-c/11 Version 11-01-01
TCIO

```
48
49 ;
50 ; tiny-c/11 subroutines
51 ;
52 .GLOBL TINYC, ADDRVA, ALPHA, ALPHAN, ASGN, ATOI, BLANKS
53 .GLOBL CANON, CEQ, CEQN, CONST, ENTER, ESET
54 .GLOBL EQ, EXPR, FACTOR, FUNDON, LINK, LIT
55 .GLOBL MC, MOVN, NEWFUN, NEWVAR, NUM, POP, PUSH, RELN, REM
56 .GLOBL RETURN, SAVE, SETARG, SKIP, SKIPST, ST, SYMNAM
57 .GLOBL TCTOI, TERM, TOPTOI, VALLOC
58 ;
59 ; tiny-c/11 installation vector entries
60 ;
61 .GLOBL INCH, OUTCH, FLOAD, FOPEN, FREAD, FWRITE, FCLOSE
62 .GLOBL USERMC, PRBEGN, STBEGN, PRDONE
63 .GLOBL BSTACK, ESTACK, BVAR, EVAR, BFUN, EFUN, BPR, EPR, MSTACK
64 ;
65 ; tiny-c/11 register assignments
66 ;
67 R0=%^00
68 R1=%^01
69 R2=%^02
70 R3=%^03
71 R4=%^04
72 R5=%^05
73 R6=%^06
74 R7=%^07
75 SP=%^06
76 PC=%^07
    .LIST
```


tiny-c/11 Version 11-01-01
TCIO

```

78                                     .TITLE tiny-c/11
79                                     .SBTTL TCIO
80                                     ;
81                                     ; tiny-c/11 input/output programmed requests
82                                     ;
83                                     .MCALL ..V2...TTYOUT,.TTYIN
84                                     .MCALL .LOOKUP,.ENTER,.WRITW,.READW,.CLOSE
85                                     ;
86                                     ; tiny-c/11 input/output routines for RT-11
87                                     ;
88 000000                                ..V2..
                                .MCALL ...CM1,...CM2,...CM3,...CM4
                                ...V2=1
89 000001                                .GLOBL GETCHR,PUTCHR,LOAD,OPEN,GETBL,PUTBL,CLOSE
90                                     ;
91                                     ; return a character from the console terminal in r0
92                                     ;
93 000000'                                .CSECT TCIO
94 000000 052737 050000 000044 GETCHR: BIS #50000,@#44
95 000006                                1$: .TTYIN
                                EMT ^O340
                                BCS -2
                                .IIF NB <>, MOVB %0,
96 000012 122700 000004                                CMPB #4,R0
97 000016 001002                                BNE 2$
98 000020 005000                                CLR R0
99 000022 000403                                BR 3$
100 000024 122700 000015 2$: CMPB #15,R0
101 000030 001766                                BEQ 1$
102 000032 042737 050000 000044 3$: BIC #50000,@#44
103 000040 000207                                RTS PC
104                                     ;
105                                     ; send the character in 2(sp) to the console terminal
106                                     ;
107 000042 052737 040000 000044 PUTCHR: BIS #40000,@#44
108 000050 122766 000012 000002                                CMPB #12,2(SP)
109 000056 001004                                BNE 4$
110 000060 012700 000015                                MOV #15,R0
111 000064                                .TTYOUT
                                .IIF NB <>, MOVB ,%0
                                EMT ^O341
                                BCS -2
112 000070 016600 000002 4$: MOV 2(SP),R0
113 000074                                .TTYOUT
                                .IIF NB <>, MOVB ,%0
                                EMT ^O341
                                BCS -2
                                000074 104341
                                000066 103776
114 000100 042737 040000 000044 BIC #40000,@#44
115 000106 000207                                RTS PC
```


tiny-c/11 Version 11-01-01
TCIO

```

116 ;
117 ; load tcload.tcf into the program space
118 ;
119 000110 004567 000000G LOAD: JSR R5,SAVE
120 000114 .LOOKUP #CSW,#1,#TCLOAD
      .IF DF ...V1
      .IIF NB <#1>, MOV #1,%0
      EMT ^O<20+#CSW>
      .IFF
      ...CM1 <#CSW>,1,<#1>
      .IF NB #CSW
      MOV #CSW,%0
      MOVB #1,1(0)
      .ENDC
      .IF NB #1
      .IF IDN <#1>,<#0>
      CLRB (0)
      .IFF
      MOVB #1,(0)
      .ENDC
      .ENDC
      ...CM2 <#TCLOAD>,2.
      .IIF NB <#TCLOAD>, MOV #TCLOAD,2.(0)
      .IIF NB <>, EMT ^O375
      .IF B
      CLR 4.(0)
      .IFF
      MOV ,4.(0)
      .ENDC
      EMT ^O375
      .ENDC
121 000146 103005 BCC 5$
122 000150 012716 000031 MOV #LOADERR,(SP)
123 000154 004737 000000G JSR PC,@#ESET
124 000160 000464 BR 11$
125 000162 005001 5$: CLR R1
126 000164 6$: .READW #CSW,#1,#BUFFER,#256.,R1
      .IF DF ...V1
      .IIF NB <#256.>, MOV #256.,%0
      CLR -(6.)
      MOV #BUFFER,-(6.)
      MOV #1,-(6.)
      EMT ^O<200+#CSW>
      .IFF
      ...CM4 <#CSW>,<#1>,<#BUFFER>,<#256.>,<R1>,#0,8.
      ...CM1 <#CSW>,<8.>,<#1>
      .IF NB #CSW
      MOV #CSW,%0

```


tiny-c/11 Version 11-01-01
TCIO

```

000170 112760 000010 000001          MOVB      #8.,1(0)
                                .ENDC
                                .IF NB #1
                                .IF IDN <#1>,<#0>
                                CLRB      (0)
                                .IFF
                                MOVB      #1,(0)
000176 112710 000001          .ENDC
                                .ENDC
000202          ...CM2 <R1>,2.
000202 010160 000002          .IIF NB <R1>,      MOV      R1,2.(0)
                                .IIF NB <>,      EMT      ^O375
000206          ...CM2 <#BUFFER>,4.
000206 012760 001544' 000004          .IIF NB <#BUFFER>,      MOV      #BUFFER,4.(0)
                                .IIF NB <>,      EMT      ^O375
000214          ...CM2 <#256.>,6.
000214 012760 000400 000006          .IIF NB <#256.>,      MOV      #256.,6.(0)
                                .IIF NB <>,      EMT      ^O375
000222          ...CM2 <#0>,8.,X
000222 012760 000000 000010          .IIF NB <#0>,      MOV      #0,8.(0)
000230 104375          .IIF NB <X>,      EMT      ^O375
                                .ENDC
127 000232 103422          BCS      9$
128 000234 005201          INC      R1
129 000236 012702 001544'          MOV      #BUFFER,R2
130 000242 006300          ASL      R0
131 000244 122712 000015          7$:    CMPB     #15,@R2
132 000250 001407          BEQ      8$
133 000252 122712 000000          CMPB     #00,@R2
134 000256 001404          BEQ      8$
135 000260 111277 000000G          MOVB     @R2,@CURSOR
136 000264 005267 000000G          INC      CURSOR
137 000270 005300          8$:    DEC      R0
138 000272 003734          BLE      6$
139 000274 005202          INC      R2
140 000276 000762          BR       7$
141 000300 105737 000052          9$:    TSTB     @#52
142 000304 001405          BEQ      10$
143 000306 012716 000031          MOV      #LOADERR,(SP)
144 000312 004737 000000G          JSR      PC,@#ESET
145 000316 000405          BR       11$
146 000320 016767 000000G 000000G 10$:    MOV      CURSOR,PROGEND
147 000326 005367 000000G          DEC      PROGEND
148 000332          11$:    .CLOSE   #1
                                .IF DF ...V1
                                EMT      ^O<160+#1>
                                .IFF
000332          ...CM3 <#1>,6.

```


tiny-c/11 Version 11-01-01
TCIO

000332	012700	003000			MOV	#6.*^0400,%0
000336	152700	000001		.IIF NB <#1>,	BISB	#1,%0
000342	104374				EMT	^0374
				.ENDC		
149	000344	000167	000000G	JMP	RETURN	
150						
151						
152						
153	000350	004567	000000G	OPEN:	JSR	R5,SAVE
154	000354	016503	000012		MOV	12(R5),R3
155	000360	003004			BGT	11\$
156	000362	012700	177777	10\$:	MOV	#-1,R0
157	000366	000167	000000G		JMP	RETURN
158	000372	022703	000020	11\$:	CMP	#20,R3
159	000376	002771			BLT	10\$
160	000400	062765	000777 000010		ADD	#511.,10(R5)
161	000406	116504	000011		MOVB	11(R5),R4
162	000412	006204			ASR	R4
163	000414	016516	000006		MOV	6(R5),(SP)
164	000420	004737	001274'		JSR	PC,@#CRAD50
165	000424	010067	001014		MOV	R0,FILEN0
166	000430	016516	000006		MOV	6(R5),(SP)
167	000434	062716	000004		ADD	#4,(SP)
168	000440	004737	001274'		JSR	PC,@#CRAD50
169	000444	010067	000776		MOV	R0,FILEN1
170	000450	016516	000006		MOV	6(R5),(SP)
171	000454	062716	000007		ADD	#7,(SP)
172	000460	004737	001274'		JSR	PC,@#CRAD50
173	000464	010067	000760		MOV	R0,FILEN2
174	000470	016516	000006		MOV	6(R5),(SP)
175	000474	062716	000013		ADD	#11.,(SP)
176	000500	004737	001274'		JSR	PC,@#CRAD50
177	000504	010067	000742		MOV	R0,FILEN3
178	000510	022765	000001 000004		CMP	#1,4(R5)
179	000516	001024			BNE	12\$
180	000520				.LOOKUP	#CSW,R3,#FILEN0
				.IF DF ...V1		
				.IIF NB <R3>,	MOV	R3,%0
					EMT	^0<20+#CSW>
000520				.IFF		
				...CM1 <#CSW>,1,<R3>		
				.IF NB #CSW		
000520	012700	001464'			MOV	#CSW,%0
000524	112760	000001 000001			MOVB	#1,1(0)
				.ENDC		

tiny-c/11 Version 11-01-01
TCIO

```

                                .IF NB R3
                                .IF IDN <R3>,<#0>
                                CLR B      (0)
                                .IFF
                                MOV B      R3,(0)
                                .ENDC
                                .ENDC
                                ...CM2 <#FILEN0>,2.
000534 012760 001444' 000002 .IIF NB <#FILEN0>,
                                .IIF NB <>,      EMT      ^O375      #FILEN0,2.(0)
                                .IF B
                                CLR      4.(0)
                                .IFF
                                MOV      ,4.(0)
                                .ENDC
                                EMT      ^O375
                                .ENDC
181 000550 103437      BCS      13$
182 000552 012763 177777 001504'      MOV      #-1,BLOCK(R3)
183 000560 070027 001000      MUL      #512.,R0
184 000564 000167 000000G      JMP      RETURN
185 000570 022765 000002 000004 12$:  CMP      #2,4(R5)
186 000576 001024      BNE      13$
187 000600      .ENTER      #CSW,R3,#FILEN0,R4
                                .IF DF ...V1
                                MOV      R3,%0
                                .IF B #FILEN0
                                CLR      -(6.)
                                .IFF
                                MOV      #FILEN0,-(6.)
                                .ENDC
                                EMT      ^O<40+#CSW>
                                .IFF
                                ...CM1 <#CSW>,2.,<R3>
                                .IF NB #CSW
                                MOV      #CSW,%0
                                MOV B      #2.,1(0)
                                .ENDC
                                .IF NB R3
                                .IF IDN <R3>,<#0>
                                CLR B      (0)
                                .IFF
                                MOV B      R3,(0)
                                .ENDC
                                .ENDC
                                ...CM2 <#FILEN0>,2.
000614 012760 001444' 000002 .IIF NB <#FILEN0>,
                                .IIF NB <>,      EMT      ^O375      #FILEN0,2.(0)

```


tiny-c/11 Version 11-01-01
TCIO

000732			...	CM2 <R4>,2.		
000732	010460	000002		.IIF NB <R4>,	MOV	R4,2.(0)
				.IIF NB <>,	EMT	^O375
000736			...	CM2 <#BUFFER>,4.		
000736	012760	001544'	000004	.IIF NB <#BUFFER>,	MOV	#BUFFER,4.(0)
				.IIF NB <>,	EMT	^O375
000744			...	CM2 <#256.>,6.		
000744	012760	000400	000006	.IIF NB <#256.>,	MOV	#256.,6.(0)
				.IIF NB <>,	EMT	^O375
000752			...	CM2 <#0>,8.,X		
000752	012760	000000	000010	.IIF NB <#0>,	MOV	#0,8.(0)
000760	104375			.IIF NB <X>,	EMT	^O375
				.ENDC		
205	000762	103417		BCS	17\$	
206	000764	005000		CLR	R0	
207	000766	012702	001544'	MOV	#BUFFER,R2	
208	000772	122712	000004	15\$: CMPB	#4,(R2)	
209	000776	001407		BEQ	16\$	
210	001000	111211		MOVB	(R2),(R1)	
211	001002	005200		INC	R0	
212	001004	005201		INC	R1	
213	001006	005202		INC	R2	
214	001010	022700	001000	CMP	#512.,R0	
215	001014	003366		BGT	15\$	
216	001016	000167	000000G	16\$: JMP	RETURN	
217	001022	012700	177777	17\$: MOV	#-1,R0	
218	001026	105737	000052	TSTB	@#52	
219	001032	001402		BEQ	18\$	
220	001034	012700	177776	MOV	#-2,R0	
221	001040	000167	000000G	18\$: JMP	RETURN	
222						
223						
224						
225	001044	004567	000000G	PUTBL: JSR	R5,SAVE	
226	001050	162706	000004	SUB	#4,SP	
227	001054	012765	000004	MOV	#4,-12(R5)	177766
228	001062	016501	000004	MOV	4(R5),R1	
229	001066	016502	000006	MOV	6(R5),R2	
230	001072	016503	000010	MOV	10(R5),R3	
231	001076	160102		SUB	R1,R2	
232	001100	062702	000001	ADD	#1,R2	
233	001104	022702	001000	CMP	#512.,R2	
234	001110	003412		BLE	19\$	
235	001112	010104		MOV	R1,R4	
236	001114	060204		ADD	R2,R4	
237	001116	010465	177770	MOV	R4,-10(R5)	
238	001122	111465	177766	MOVB	(R4),-12(R5)	
239	001126	112714	000004	MOVB	#4,(R4)	

tiny-c/11 Version 11-01-01
TCIO

240 001132 062702 000002
241 001136 006202
242 001140 005263 001504'
243 001144 016304 001504'
244 001150

19\$: ADD #2,R2
ASR R2
INC BLOCK(R3)
MOV BLOCK(R3),R4
.WRITW #CSW,R3,R1,R2,R4
.IF DF ...V1
.IIF NB <R2>, MOV R2,%0
CLR -(6.)
MOV R1,-(6.)
MOV R3,-(6.)
EMT ^O<220+#CSW>

001150
001150

.IFF
...CM4 <#CSW>,<R3>,<R1>,<R2>,<R4>,<#0>,9.
...CM1 <#CSW>,<9.>,<R3>
.IF NB #CSW

001150 012700 001464'
001154 112760 000011 000001

MOV #CSW,%0
MOVB #9.,1(0)

.ENDC
.IF NB R3
.IF IDN <R3>,<#0>

CLRB (0)

001162 110310

.IFF

MOVB R3,(0)

.ENDC

001164
001164 010460 000002

.ENDC
...CM2 <R4>,2.
.IIF NB <R4>, MOV R4,2.(0)
.IIF NB <>, EMT ^O375

001170
001170 010160 000004

...CM2 <R1>,4.
.IIF NB <R1>, MOV R1,4.(0)
.IIF NB <>, EMT ^O375

001174
001174 010260 000006

...CM2 <R2>,6.
.IIF NB <R2>, MOV R2,6.(0)
.IIF NB <>, EMT ^O375

001200
001200 012760 000000 000010
001206 104375

...CM2 <#0>,8.,X
.IIF NB <#0>, MOV #0,8.(0)
.IIF NB <X>, EMT ^O375
.ENDC

245 001210 103413
246 001212 006300
247 001214 122765 000004 177766
248 001222 001404
249 001224 016504 177770
250 001230 116514 177766
251 001234 000167 000000G
252 001240 012700 177777
253 001244 000763

BCS 22\$
ASL R0
20\$: CMPB #4,-12(R5)
BEQ 21\$
MOV -10(R5),R4
MOV -12(R5),(R4)
21\$: JMP RETURN
22\$: MOV #-1,R0
BR 20\$

tiny-c/11 Version 11-01-01
TCIO

```

254
255
256
257 001246 004567 000000G
258 001252 016503 000004
259 001256

001256
001256 012700 003000
001262 150300
001264 104374

260 001266 005000
261 001270 000167 000000G
262
263
264
265 001274 005000
266 001276 122776 000040 000002
267 001304 001414
268 001306 117601 000002
269 001312 162701 000022
270 001316 022701 000050
271 001322 003002
272 001324 162701 000116
273 001330 070127 003100
274 001334 060100
275 001336 005266 000002
276 001342 122776 000040 000002
277 001350 001414
278 001352 117601 000002
279 001356 162701 000022
280 001362 022701 000050
281 001366 003002
282 001370 162701 000116
283 001374 070127 000050
284 001400 060100
285 001402 005266 000002
286 001406 122776 000040 000002
287 001414 001412
288 001416 117601 000002
289 001422 162701 000022
290 001426 022701 000050
291 001432 003002
292 001434 162701 000116
293 001440 060100
294 001442 000207

```

```

;
; close unit 2(sp)
;
CLOSE: JSR R5,SAVE
MOV 4(R5),R3
.CLOSE R3
.IF DF ...V1
EMT ^O<160+R3>
.IFF
...CM3 <R3>,6.
MOV #6.*^O400,%0
.IIF NB <R3>, BISB R3,%0
EMT ^O374
.ENDC
CLR R0
JMP RETURN
;
; convert 3 ascii characters at 2(sp) to RAD50 representation in R0
;
CRAD50: CLR R0
CMPB #40,@2(SP)
BEQ 24$
MOVB @2(SP),R1
SUB #22,R1
CMP #50,R1
BGT 23$
SUB #116,R1
MUL #3100,R1
23$: ADD R1,R0
24$: INC 2(SP)
CMPB #40,@2(SP)
BEQ 26$
MOVB @2(SP),R1
SUB #22,R1
CMP #50,R1
BGT 25$
SUB #116,R1
25$: MUL #50,R1
ADD R1,R0
26$: INC 2(SP)
CMPB #40,@2(SP)
BEQ 28$
MOVB @2(SP),R1
SUB #22,R1
CMP #50,R1
BGT 27$
SUB #116,R1
27$: ADD R1,R0
28$: RTS PC

```


Crunched Program Preparation System
Version 1.00

Appendix C -- Crunched PPS Code
=====

```
putchar char c
[if(c==0)c='"'
return MC c,1
]
getchar
[return MC 2
]
chrdy[return MC 12]
pft char f(0),t(0)[MC f,t,13]
gs char b(0)
[int l
while((b(1)=MC(2))!=13)[
if(b(1)==24)[l=0;pl""]
else if (b(1)==127)[if(l>0)l=l-1]
else l=l+1
]
b(1)=0
return l
]
ps char b(0)[
int k(0);k(0)=1
pft(b,b-1+scann(b,b+30000,0,k))
]
pl char b(0)
[MC 13,1
ps b
]
alpha char a
[
if((a>='a')*(a<='z'))return 1
if((a>='A')*(a<='Z'))return 1
]
num char b(5)
int v(0)
[int k
v(0)=0
while(k<5)
[if((b(k)<'0')+(b(k)>'9'))return k
v(0)=10*v(0)+b(k)-'0'
k=k+1
]
return k
]
```


2

Crunched Program Preparation System
Version 1.00

```
atoi char b(0)
int v(0)
[ int k,s
char c
s=1
c=b(0)
while((c==' ')+(c=='-')+(c=='+'))
[ if(c=='-')s=-1
c=b(k=k+1)
]
k=k+num(b+k,v)
v(0)=s*v(0)
return k
]
pn int n
[
MC ' ',1
MC n,14
]
gn
[ char b(20)
int v(0)
while(1)
[ gs b
if(atoi b,v)return v(0)
ps"number required "
]
]
ceqn char a(0),b(0)
int n
[ int k
k=-1
while((k=k+1)<n)if(a(k)!=b(k))return 0
return 1
]
```


Crunched Program Preparation System
Version 1.00

```
index char i(0)
int l
char f(0)
int n
[
if(n<=0)return 1
if(l<=0)return 0
int a,d(0)
while(a+n<=1)[
d(0)=1
a=a+1+scann(i+a,i+1-n,f(0),d)
if(d(0))return 0
if(ceqn(i+a,f+1,n-1))return a
]
]
move char a(0),b(0)
[int k
k=-1
while(a(k=k+1)!=0)b(k)=a(k)
b(k)=0
return k
]
gc
[char f
f=MC 2
while(MC(2)!=13)[
return f
]
movebl char a(0),b(0);int n
[MC(a,b,n,7)]
countch char a(0),b(0),c
[return MC(a,b,c,8)]
scann char a(0),b(0),c;int n(0)
[return MC(a,b,c,n,9)]
```


4
Crunched Program Preparation System
Version 1.00

```
readfile char n(0),w(0),l(0)
int u
[ int k,t
MC(1,n,0,u,3)
while(1)
[
k=MC(w,u,4)
if(k== -1) return t
if(k< -1) return k
t=t+k
if((w=w+k)>1)
[pl"Too big"
return -2
]
]
]
writefile char n(0),b(0),e(0)
int u
[ int k,t,1
MC(2,n,e-b+1,u,3)
while(b<=e)
[
l=e-b
if(l>255) l=255
k=MC(b,b+1,u,5)
if(k<0) return k
if(k>0) return -k
t=t+l+1
b=b+l+1
]
k=MC(u,6)
if(k<0) return k
if(k>0) return -k
return t
]
fopen int m;char n(0);int s,u[return MC m,n,s,u,3]
fread char a(0);int u[return MC a,u,4]
fwrite char f(0),t(0); int u[return MC f,t,u,5]
fclose int u[MC u,6]
endlibrary
```


Crunched Program Preparation System
Version 1.00

```
int er(0),cu,lo,pe,lp
int ll, la
char ft(20),tt(20)
int fl,tl
char ln(64),pr( 2000)
main
[char c
int v(1)
lp= 2000
pl(0)=13 pr(0)=13
while(1)
[ps ">"
while((ll=gs(ln))==0)[ ]
c=ln(0)
if(c=='.')
[if(num(ln+1,v))go(v)
else if((ln(2)==0)+(alpha(ln(2))==0))
[c=ln(1)
if(c=='p')pt
else if(c=='d')dl
else if(c=='l')oi
else if(c=='c')ch
else if(c=='/')fa
else if(c=='r')gi
else if(c=='w')gu
else if(c=='x')return
else [ps"???";pl""]
]else st
]
else if(c=='-')up
else if(c=='+')do
else in
]
]
```


6

Crunched Program Preparation System
Version 1.00

```
pi int n
[ int f,l,v(0)
v(0)=n
f=fc
lo=lo+v(0)-1
l=cu+scann(pr+cu,pr+pe,13,v)
cu=1
lo=lo-v(0)
MC pr+f,pr+1,13
]
fc
[ int k
if((k=cu)==0)return 0
while(pr(k=k+1)!=13)if(k<0)break
return k+1
]
lc
[ int k
k=cu-1
while(pr(k=k+1)!=13)if(k>=pe)break
return k
]
nl
[ if((cu=lc()+1)>pe)
[ cu=pe
return 0
]
return lo=lo+1
]
bl
[ if((cu=fc()-1)<0)cu=0
else lo=lo-1
]
pt[
int v(0)
if(ln(2))num(ln+3,v)
else v(0)=1
pi(v(0))
]
```


Crunched Program Preparation System
Version 1.00

```
dl
[ int f,l,v(1)
if(cu==0)
[ps"cannot delete line 0";pl""
return
]
if(ln(2)==0)v(0)=1
else num(ln+3,v)
la=la-v(0)
f=fc
l=cu+scann(pr+cu,pr+pe,13,v)
la=la+v(0)
lo=lo-1
cu=f-1
if(l<pe)movebl(pr+l+1,pr+pe,-(l-f+1))
pe=pe-(l-f+1)
]
oi
[
int k
if(ln(2)!=0)
[fl=move(ln+3,ft)
if(ft(0)=='^')ft(0)=13
if(ft(fl-1)=='^')ft(fl-1)=13
]
if(fl==0)
[pl"locate what?";pl""
return
]
if(nonl()!=0)[
if(k=index(pr+cu-1,pe-cu+2,ft,fl)) [
cu=cu-2+k
if(pr(cu)==13)cu=cu+1
lo=countch(pr,pr+cu-1,13)
pi 1
]
else[ps"?";pl""]
]
else[pl"at bottom";pl""]
]
```


8

Crunched Program Preparation System
Version 1.00

```
ch[
char d
int p,f,l
if(ln(2)!=0)[
d=ln(2)
p=2
while(ln(p=p+1)!=d)[
if(ln(p)==0)[
ln(p+1)=0
break
]
]
ln(p)=0
fl=move(ln+3,ft)
tl=move(ln+p+1,tt)
if(tl)if(tt(tl-1)==d)tl=tl-1
]
f=fc
l=lc()-1
int k
if(k=index(pr+f,l-f+1,ft,fl)) [
cu=f+k-1
movebl(pr+cu+fl,pr+pe,tl-fl)
pe=pe+tl-fl
if(tl)movebl(tt,tt+tl-1,pr+cu-tt)
]
pi 1
]
in
[
ll==ll+1
if(pe+ll>lp)
[ps"won't fit";pl""
return
]
if(nl)movebl(pr+cu,pr+pe,ll)
else[cu=cu+1;lo=lo+1]
pe=pe+ll
movebl(ln,ln+ll-1,pr-ln+cu)
pr(cu+ll-1)=13
la=la+1
]
```


Crunched Program Preparation System
Version 1.00

```
wh
[ int f,l,u,b
pn lo;ps" --- err ";pn er(0);pl""
u=cu
f=fc
b=u-f
l=lc
f=f-1
while((f=f+1)<1)putchar(pr(f));pl""
while((b=b-1)>=0)putchar(' ')
putchar '<';pl""
]
do
[ int v(1)
if(ln(1)==0)v(0)=1
else num(ln+1,v)
lo=lo+v(0)
v(0)=v(0)+1
cu=cu+scann(pr+cu,pr+pe,13,v)
lo=lo-v(0)
pi(1)
]
up
[ int l,v(1)
if(ln(1)==0)l=1
else[num(ln+1,v);l=v(0)]
while((l=l-1)>=0)bl
pi(1)
]
go int l(1)
[ lo=l(0)
l(0)=l(0)+1
cu=scann(pr,pr+pe,13,l)
lo=lo-l(0)
pi(1)
]
fa
[pn lo;pn la;pn pe;pn lp-pe;pl""]
```


Crunched Program Preparation System
Version 1.00

```
st
[while(ll<=64)
[ln(ll)=' '
ll=ll+1
]
MC(er,ln+1,pr+pe,pr,11)
if(cu<0)cu=0;if(cu>pe)cu=pe
lo=countch(pr,pr+cu-1,13)
pl"";pl""
if(er(0))
if(er(0)==99)[ps"stopped";pl""]
else wh
]
gi
[int k
k=readfile(ln+3,pr+pe+1,pr+lp,1)
if(k<0)return
pe=pe+k
la=countch(pr+1,pr+pe,13)
pn k;pl"";fa
]
gu
[fa
pn writefile(ln+3,pr+1,pr+pe,1)
pl""
]
```


INDEX

- ADDRVAL
 - defined.....5-10
 - use of.....5-8
- alpha
 - defined.....2-23
- ALPHA
 - defined.....5-28
- ALPHANUM
 - defined.....5-29
- Alternation
 - as a compound if statement...2-19
- Application Level
 - and APPLVL.....4-31
 - defined.....4-30
 - in MC 11.....5-27
- APPLVL
 - and MC 11.....5-27
 - explained.....4-31
- Arguments
 - as arrays.....2-11
 - as used in MCs.....2-30
 - correct number of.....2-10
 - in interpreter subroutines....5-1
 - local copies.....2-13
 - of PPS commands.....3-6
 - use of.....2-3
 - values of.....1-5, 2-10
- Array
 - as an argument.....2-11
 - defined.....2-2
 - used as matrix.....2-3
 - use of.....2-2+
 - with size 0.....2-6
- ASGN
 - action of.....5-19
 - calling relationship.....5-18
 - characters used.....5-20
 - defined.....5-21
 - called by ST.....5-21
 - statement analyzer.....5-17
- Assignment
 - defined.....2-8
 - multiple.....2-10
 - use of.....2-8
- atoi
 - defined.....2-23
 - notes on.....2-27
- ATOI
 - defined.....5-29
- BFUN
 - and function table.....5-8
 - and layer 0.....5-8
- blanks
 - defined.....4-2
- BLANKS
 - defined.....5-17
 - scanning tool.....5-14
- Blanks
 - and scanning tools.....5-15+
 - in function names.....2-1
 - in Optional Library
 - in function blanks.....4-2
 - in function htoi.....4-2
 - in PPS
 - in change command.....3-5
 - in locate command.....3-5
 - to separate text strings...3-4
 - in Standard Library
 - in function atoi.....2-23
 - in function num.....2-23
 - in function pn.....2-22
 - in variable names.....2-1
 - to print a blank line.....1-4
- Brackets
 - consistency in use of.....1-13
 - use of.....1-12
- break
 - defined.....2-18

BSTACK		chrdy	
and tiny-c stack.....	5-10	as MC 12.....	5-28
BVAR		defined.....	2-24
and variable table.....	5-4	Commas	
BZAP		to separate arguments.....	2-11
defined.....	5-29	Comment	
Calling A Function		defined.....	1-2, 2-1
defined.....	2-10	Compound Statement	
CANON		as a PPS command.....	3-6
defined.....	5-10	defined.....	1-3
use of.....	5-8	examined in detail.....	1-8+
Carriage Return		in structured programming.....	1-7
and line delete.....	3-2	nesting of.....	1-10
and line zero.....	3-2	illustrated.....	1-11+
and text change.....	3-5	use of.....	1-10
in library function gc.....	2-26	CONST	
in library function gs.....	2-21	and statement analyzer.....	5-20
in Piranha Game commands.....	4-6	defined.....	5-16
Case		scanning tool.....	5-14
see Alternation		Constants	
ceq		as pointers.....	2-6
defined.....	4-2	character.....	2-5
illustrated.....	4-3	integer.....	2-5
notes on.....	4-4	Copying A File	
CEQ		notes on.....	4-34
defined.....	5-29	program, illustrated.....	4-35
ceqn		counch	
defined.....	2-23	as MC 8.....	5-26
illustrated.....	4-20	defined.....	2-24
notes on.....	2-27, 4-30	illustrated.....	4-21
CEQN		notes on.....	2-27, 4-30
defined.....	5-30	CTOI	
Change		defined.....	5-30
as PPS command.....	3-4+	CURFUN	
notes on using.....	3-8	and function table.....	5-8
char		explained.....	5-7
and data objects.....	5-2	CURGLBL	
declaration statement.....	2-2	and function table.....	5-8
object size.....	5-3	Current Line	
Character Constants		as a new text line.....	3-3
defined.....	2-5	how to print.....	3-3
		CURSOR	
		and scanning tools.....	5-14
		defined.....	5-14+
		in MC 11.....	5-27
		in subroutine ESET.....	5-30

Data Objects		ESET	
properties of.....	5-2	defined.....	5-30
Data Types		and tiny-c errors.....	5-1
introduced.....	2-2	ESTACK	
Delete		and tiny-c stack.....	5-10
and line zero.....	3-7	Evaluation	
as PPS command.....	3-4, 3-7	of assignment series.....	2-10
standard character DEL.....	3-1	of expressions.....	2-9
modification of.....	3-1	EVAR	
standard line DEL.....	3-2	and variable table.....	5-4
EFUN		EXPR	
and function table.....	5-8	action of.....	5-19
else		calling relationship.....	5-18
defined.....	2-17	characters used.....	5-20
endlibrary		defined.....	5-22
in subroutine LINK.....	5-24	called by RELN.....	5-19
Enter an Application Program		statement analyzer.....	5-17
see MC 11		Expressions	
ENTER		defined.....	2-5
action of.....	5-19	used as pointers.....	2-15
calling relationship.....	5-18		
characters used.....	5-20	FACTOR	
defined.....	5-23	action of.....	5-19
called by FACTOR.....	5-19, 5-22	calling relationship.....	5-18
statement analyzer.....	5-17	characters used.....	5-20
EPR		defined.....	5-22
use of.....	5-10	called by TERM.....	5-19
EQ		statement analyzer.....	5-17
defined.....	5-14	fclose	
called by ASGN.....	5-19, 5-21	as MC 6.....	5-25
use of.....	5-12	defined.....	2-25
ERR		notes on.....	2-28
in MC 11.....	5-27	FNAME	
in subroutine ESET.....	5-30	in subroutine CONST.....	5-16
ERRAT		in subroutine FACTOR.....	5-21
in subroutine ESET.....	5-30	in subroutine SYMNAME.....	5-15
Error Numbers		in subroutine VALLOC.....	5-21
explained.....	3-10	fopen	
Escape		as MC 3.....	5-25
character defined.....	6-22	defined.....	2-25
in PPS.....	3-6	notes on.....	2-28
to terminate applications....	4-31	fread	
use in level zero programs...4-31		as MC 4.....	5-25
		defined.....	2-25
		notes on.....	2-28

Functions

- active.....5-5
- and their variables.....2-3+
- examined in detail.....2-10+
- in the variable table.....5-4
- introduced.....1-14

Function Names

- duplication of.....2-4
- in structured programming.....1-7
- in the variable table.....5-5
- restrictions on.....2-1
- used as a variable.....2-6

Function Number

- see Machine Call

Function Table

- explained.....5-8
- in subroutine ENTER.....5-23
- subroutines for.....5-8

FUNDONE

- defined.....5-9
- called by ENTER.....5-23
- use of.....5-8

fwrite

- as MC 5.....5-25
- defined.....2-25
- notes on.....2-28

gc

- defined.....2-22
- notes on.....2-26

getchar

- defined.....2-22
- notes on.....2-26

GETCHAR

- defined.....5-25

Global Variables

- in structured programming.....1-8
- use of.....2-4
- values of.....1-6, 2-4

gn

- argument number.....2-11
- defined.....2-22
- use of.....1-4

gs

- defined.....2-21
- notes on.....2-26

htoi

- defined.....4-2

if

- defined.....2-17
- example of.....1-4
- in compound statements.....1-8+

Indenting

- choice of styles.....1-13
- consistency in.....1-13
- versus tabs.....3-2

index

- defined.....2-24
- illustrated.....4-26, 4-27
- notes on.....2-27, 4-30

int

- and data objects.....5-2
- declaration statement.....2-2
- object size.....5-3

Integer Constants

- defined.....2-5

Interrupt

- defined.....5-26

itoa

- defined.....4-2
- notes on.....4-5

itoh

- defined.....4-2
- notes on.....4-4

Level

- see Application Level

Libraries

- defined.....2-19

Library Symbols

- in function table.....5-8

Line Feed

- see Carriage Return

Line Numbers

- before text lines.....3-2
- in program buffer.....3-3+
- notes on.....3-7

Line Zero

- defined.....3-2
- deleting of.....3-7

LINK		
action of.....	5-19	
in MC 11.....	5-27	
statement analyzer.....	5-17	
LIT		
and statement analyzer.....	5-20	
defined.....	5-15	
scanning tool.....	5-14	
LNAME		
in subroutine CONST.....	5-16	
in subroutine FACTOR.....	5-21	
in subroutine SYMNAME.....	5-15	
in subroutine VALLOC.....	5-21	
Local Variables		
examined in depth.....	1-16	
illustrated.....	1-17	
use of.....	2-4	
values of.....	1-6, 2-4	
versus global.....	1-8	
Locate		
as PPS command.....	3-4	
notes on using.....	3-7+	
Loop		
see while		
Lvalue		
defined.....	5-11	
Machine Call		
explained in detail.....	2-30	
function number.....	2-30	
how to name.....	2-21	
in PPS commands.....	3-6	
private, defined.....	2-20	
standard, defined.....	2-20	
standard functions defined...	5-24	
use of.....	2-21	
writing your own.....	2-29+	
MC		
see Machine Call		
MC 11		
defined.....	5-26	
use of.....	4-30	
MCARGS		
defined.....	2-32	
MCESET		
defined.....	2-31	
Morse Code Generator		
instructions.....	4-31	
notes on.....	4-34	
program, illustrated.....	4-32	
move		
defined.....	2-24	
movebl		
as MC 7.....	5-26	
defined.....	2-24	
illustrated.....	4-27	
notes on.....	2-27, 4-30	
moven		
defined.....	4-2	
MOVN		
defined.....	5-30	
NEWFUN		
defined.....	5-9	
called by ENTER.....	5-19, 5-23	
use of.....	5-8	
New Line		
see Carriage Return		
NEWVAR		
defined.....	5-9	
called by VALLOC.....	5-19, 5-21	
use of.....	5-8	
num		
defined.....	2-23	
notes on.....	2-27	
pointers as arguments.....	2-15	
NUM		
defined.....	5-31	
Numbers		
range.....	1-6, 2-34	
Operators		
defined.....	2-6	
precedence.....	2-9	
used in tiny-c.....	2-7	
Optional Library		
defined.....	2-20	
functions explained.....	4-1	
functions illustrated.....	4-3	

- Parentheses
 - removal of.....2-12
 - to change precedence.....2-10
 - to enclose arguments...2-11, 2-13
- Personal Library
 - defined.....2-20
- pft
 - as MC 13.....5-28
 - defined.....2-25
- Piranha Fish
 - instructions.....4-5
 - notes on.....4-17
 - program, illustrated.....4-9+
- pl
 - argument to.....2-11
 - defined.....2-22
 - use of.....1-3
- pn
 - as MC 14.....5-28
 - defined.....2-22
- Pointers
 - arrays used as.....2-6, 2-11
 - as character string.....2-15
 - as expressions.....2-14
 - as variables.....2-14
 - defined.....2-14
 - in MC 11.....5-26
 - in Standard library
 - functions.....2-27
- PONE
 - defined.....5-13
 - use of.....5-12
- POP
 - defined.....5-13
 - use of.....5-12
- POPTWO
 - defined.....5-14
 - use of.....5-12
- Portability View
 - defined.....2-29
- PPS
 - commands explained.....3-3+
 - commands that "bump".....3-6
 - defined.....3-1
 - detects errors.....3-9
- PPS, cont.
 - error numbers defined.....3-10
 - program, illustrated.....4-18+
 - notes on program.....4-30
 - use of, illustrated.....3-11
- PPS Commands
 - .p.....3-3
 - .d.....3-4
 - +n.....3-4
 - n.....3-4
 - .n.....3-4
 - .l.....3-4
 - .c.....3-4
 - ./.....3-5
 - .r.....3-5
 - .w.....3-5
- Primaries
 - order of evaluation.....2-8
 - types of.....2-5
- Private View
 - defined.....2-28
- PROGEND
 - in subroutine SKIP.....5-15
 - use of.....5-9
- Program Buffer
 - bumping top or bottom.....3-6
 - defined.....3-2
 - inserting new text lines.....3-3
- Program Interrupt
 - introduced.....3-6
- Program Level
 - see Application Level
- Program Preparation System
 - see PPS
- PRUSED
 - use of.....5-9
- ps
 - argument number.....2-11
 - defined.....2-22
 - illustrated.....1-5
- PUSH
 - defined.....5-12
 - use of.....5-12
- PUSHK
 - defined.....5-13
 - use of.....5-12

putchar		Semi-colons	
defined.....	2-22	in a tiny-c statement.....	1-4
notes on.....	4-30	SETARG	
PUTCHAR		action of.....	5-19
defined.....	5-24	calling relationship.....	5-18
PZERO		defined.....	1-23
defined.....	5-13	called by ENTER.....	5-19, 5-23
use of.....	5-12	statement analyzer.....	5-17
random		Simple Statements	
defined.....	4-1	how to use.....	2-18
illustrated.....	1-2	types of.....	2-17
notes on.....	2-11	16-Bit Arithmetic Tools (8080)	
readfile		subroutines.....	5-32
defined.....	2-22	SKIP	
notes on.....	2-28	defined.....	5-15
RELN		scanning tool.....	5-14
action of.....	5-19	SKIPST	
calling relationship.....	5-18	action of.....	5-19
characters used.....	5-20	calling relationship.....	5-18
defined.....	5-22	characters used.....	5-20
statement analyzer.....	5-17	defined.....	5-23
REM		called by ST.....	5-21
defined.....	5-16	statement analyzer.....	5-17
scanning tool.....	5-14	Software Modularity	
Remainder		in structured programming.....	1-7
defined.....	2-7	ST	
Results		action of.....	5-19
defined.....	5-1	calling relationship.....	5-18
return		characters used.....	5-20
defined.....	2-18	defined.....	5-21
RETURN		called by ENTER.....	5-23
defined.....	5-31	statement analyzer.....	5-17
SAVE		Standard Library	
defined.....	5-31	defined.....	2-19
scann		functions explained.....	2-21
as MC 9.....	5-26	notes on.....	2-26
defined.....	2-24	Statement Analyzer	
illustrated.....	4-28	use of.....	5-17
notes on.....	2-27+, 4-30	Structured Programming	
Scanning Tools		explained.....	1-7
subroutines.....	5-15+	Subscript	
use of.....	5-14	error.....	2-6
		size.....	2-6
		use of.....	2-2
		with array of size zero.....	2-6
		Symbol	
		defined.....	5-16

SYMNAME		VALLOC	
defined.....	5-15	action of.....	5-19
scanning tool.....	5-14	calling relationship.....	5-18
statement analyzer.....	5-20	characters used.....	5-20
		defined.....	5-21
Tabs		called by SETARG.....	5-19, 5-23
see Indenting		called by ST.....	5-21
TCTOI		statement analyzer.....	5-17
defined.....	5-31	Variable Address	
TERM		defined.....	5-4
action of.....	5-19	Variable Class	
calling relationship.....	5-18	defined.....	5-3
characters used.....	5-20	length of.....	5-3
defined.....	5-22	Variable Length	
called by EXPR.....	5-19	defined.....	5-3
statement analyzer.....	5-17	Variable Names	
Text Lines		explained.....	5-2
in PPS.....	3-2	in structured programming.....	1-7
to change.....	3-4+	restrictions.....	2-1
to delete.....	3-4	Variables	
to locate.....	3-4	as pointers.....	2-6, 2-14
to print.....	3-3	class.....	5-3
Tiny-c		declaring of.....	1-3
expressions.....	1-5	defined.....	2-2
sample program walk-through...	1-1	global.....	1-4
Tiny-c Stack		local.....	1-5
explained.....	5-10+	types of declarations....	2-3, 5-2
subroutines.....	5-12	Variable Table	
TOPDIF		and active functions.....	5-5
defined.....	5-14	explained.....	5-4
use of.....	5-12	layers in.....	5-6
TOPTOI		in subroutine LINK.....	5-19, 5-24
defined.....	5-13	subroutines for.....	5-8
in writing your own MCs.....	2-32		
use of.....	5-12	while	
TV Graphics		defined.....	1-3
functions, in tiny-c.....	4-35	in compound statements.....	1-9
graphics display program,		writefile	
illustrated.....	4-38	defined.....	2-23
notes on program.....	4-36	notes on.....	2-28
		ZERO	
		defined.....	5-32



tiny c associates
post office box 269
holmdel, new jersey 07733

tiny-c
Newsletter

DATE: November, 1978
NUMBER: 1

1.0 Introduction

Tiny-c Associates will, from time to time, publish a newsletter for all registered owners of the tiny-c Owner's Manual. This is the first one. The intent is to share software ideas, bug notices and fixes, programming techniques, requests for new features and the like among tiny-c aficionados. Your participation in the Newsletter is welcome. Send your contributions to Tiny-c Associates, P. O. Box 269, Holmdel, New Jersey 07733. Be sure to indicate your contribution is for publication; if no clear indication is given, we will assume your letter is a private correspondence.

2.0 tiny-c/11 Addenda

Version 11-01-01 of tiny-c/11 appears in Appendix B of the tiny-c Owner's Manual. The fixes discussed in Sections 2.1 through 2.4 have already been incorporated into the machine-readable versions of tiny-c and apply only to this published version. However, the fixes described in Sections 2.5 and 2.6 should also be applied to the machine-readable Version 11-01-02.

2.1 Interpreter Subroutine REM (tiny-c/11-01-01)

Version 11-01-01 of REM did not begin by skipping blanks. This causes incorrect handling of comments which appear between a condition and its statement. Thus, the valid tiny-c construction

```
if(k>0) /* a comment
    pn k
```

would cause an interpreter error. More satisfactory REM code is as follows:


```
REM:  MOV    CURSOR,R1          ; save CURSOR for diagnostic
      CMPB   #' ,@CURSOR        ; skip leading blanks
      BNE    1$
      CMP    CURSOR,PROGEND      ; check for run away
      BHS    4$
      INC    CURSOR
      BR     REM
1$:   CMPB   #12,@CURSOR         ; jump over newlines
      BNE    2$
      INC    CURSOR
      BR     REM
2$:   MOV    CURSOR,R0          ; a newline is a new same
      CMPB   #' /,(R0)          ; check for a comment
      BNE    5$
      CMPB   #' *,1(R0)
      BNE    5$
      ADD    #2,CURSOR
3$:   MOV    CURSOR,R0          ; move to end of line
      INC    CURSOR
      CMPB   #12,(R0)
      BEQ    REM
      CMP    PROGEND,CURSOR      ; check for run away
      BHS    3$
4$:   MOV    R1,CURSOR          ; restore CURSOR for diagnostic
      MOV    #CURERR,-(SP)
      JSR    PC,@#ESET          ; report a cursor error
      TST    (SP)+
5$:   RTS    PC
```

2.2 Interpreter Subroutine SETARG (tiny-c/11-01-01)

The passing of a character variable to a function which declares the corresponding argument as an integer produced incorrect results in Version 11-01-01. This was due to SETARG's setting the two-byte integer value to the character byte and its adjacent byte rather than producing the second byte by extending the sign of the character byte. Better SETARG code is as follows:


```

SETARG: JSR    R5,SAVE
        MOV    4(R5),R0                ; set the argument's type
        CMPB   #'A',1(R0)
        BNE    1$
        MOV    4(R0),R2                ; if Actual, value is stuff
        BR     2$
1$:      MOV    4(R0),R3                ; else what stuff points to
        MOVB   (R3),(SP)
        MOVB   1(R3),1(SP)
        MOV    (SP),R2
2$:      TSTB   (R0)                    ; test for a one-byter
        BNE    3$
        CMPB   #1,2(R0)
        BNE    3$
        MOVB   R2,R2                  ; extend the sign
3$:      MOV    R2,(SP)                ; allocate the value
        MOVB   6(R5),R0
        MOV    R0,-(SP)
        JSR    PC,@#VALLOC
        JMP    RETURN

```

2.3 The Call to USERMC in MC (tiny-c/11-01-01)

The code which sets up the machine stack for a call to USERMC in Version 11-01-01 of MC is incorrect. The correct code is as follows:

```

        MOV    4(R5),(SP)              ; load number of arguments
        MOV    -10(R5),-(SP)           ; load function number
        SUB    #1000,.(SP)             ; subtract 1000 from it
        JSR    PC,@#USERMC            ; call USERMC

```

2.4 The In Array in FPS (tiny-c/11-01-01)

To accomodate the longer terminal input lines typical of DEC installations, the input array, in or line, in FPS should be dimensioned at 133 rather than 64.

2.5 Channel Number as Table Offset in TCIO (tiny-c/11-01-02)

There are five places in TCIO (once in OPEN, and twice each in GETBL and PUTBL) where the channel number in R3 is used to address the BLOCK table as BLOCK(R3). Since BLOCK is a word-entry table, this causes a system trap when odd-numbered channels are being accessed. The easiest fix is to do an ASL R3 before the BLOCK(R3) instructions and an ASR R3 after the instructions. Alternatively, the five instructions using BLOCK(R3) could be changed to byte instructions. We thank John Osuder of Homewood, Illinois for pointing this out.

2.6 Transient Value Changes in Interrupt Systems (tiny-c/11-01-02)

Unfortunately, the tiny-c/11 subroutine TCTOI which returns a full word from two adjacent byte addresses in memory uses the word below the word the stack pointer is pointing to form the full word. Should an interrupt occur after the word has been formed here but before it is moved to R0, the value that is actually returned is the PSW at the time of the interrupt. Obviously, this can produce some strange results and be quite maddeningly unreproducible. It is HIGHLY RECOMMENDED that the following patch be applied to all tiny-c/11-01-02 systems:

.RUN PATCH

```
FILE NAME--  
*SY:TINYC.SAV  
*11720/ 116666 116646 <LF>  
11722/ 4 <LF>  
11724/ 177776 116666 <LF>  
11726/ 116666 000004 <LF>  
11730/ 2 000001 <LF>  
11732/ 177777 012600 <LF>  
11734/ 16600 000207 <LF>  
11736/ 177776 000240 <LF>  
11740/ 207 000240 <CR>  
*E
```

This patch corresponds to the following new version of TCTOI:

```
TCTOI:  MOVB    4(SP),-(SP)  
        MOVB    4(SP),1(SP)  
        MOV     (SP)+,R0  
        RTS     PC  
        NOP  
        NOP  
        .END
```

2.7 tiny-c/11 Running under RT-11/FB (tiny-c/11-01-02)

Dr. Charles Retter of Framingham, Massachusetts notes that bit 6 in the JSW must be set if the .TTINR request in TSTCHR is to perform satisfactorily. If the bit is not set, the RT-11 monitor will block the execution of the job until a character is received, defeating the purpose of TSTCHR. Dr. Retter also notes that odd buffer addresses to .WRITW under FB will cause an odd address trap. We have not observed this problem with RT-11 V03B, and would welcome suggestions for dealing with it from other RT-11/FB users.

3.0 tiny-c/8080 Addenda

3.1 Corrected Listings (tiny-c/8080-01-01)

Three instructions on the first page of the Appendix A 8080 listings have wrong addresses. These are:

wrong version		correction	
address		address	
2000	C3 58 26	2000	C3 58 2C ✓
2003	C3 7E 26	2003	C3 7E 2C ✓
2006	C3 81 26	2006	C3 81 2C ✓

The machine-readable media are correct.

3.2 Bug Fixes (tiny-c/8080-01-01)

The contents of address 2D27 should be C4 (CNZ). The listings in Appendix A show CC (CZ) and the media may have CD (CALL).

At location 30E8, an LXI B,1 is needed before the CALL FCLOSE for some operating systems.

✓ In Chapter VI, page 6-12 (second page of Figure 6-1) the lines at 6044 should read:

✓ 6044 F8 30 0400 DW 30F8H

✓ In line 604C change the D5 to DS.

3.3 FREAD Definition Clarified (tiny-c/8080)

David McWherter, Jr., of Cornwells Heights, Pennsylvania, points out that FREAD shouldn't return an EOF unless the returned count is zero. The EOF signal is to be returned on the FREAD call after the last record of data is returned. It does not accompany the last record.

3.4 Relocater Bug

The relocater program in Figure 6-1, page 6-11, of the tiny-c Owner's Manual does not properly relocate the address at 2D3A. This address will have to be relocated manually and should be set to -BUFF-1.

4.0 PPS Errata and Addenda

✓4.1 Corrections to Appendix C--Crunched PPS Code

The following corrections should be made to the "crunched" version of PPS which appears in Appendix C of the tiny-c Owner's Manual. The version of PPS in Chapter IV and those distributed on machine-readable media are correct.

page ====	function =====	line =====	from =====	to =====
✓5	main	5	pl(0)=13	pr(0)=13
✓6	fc	4	k=k+1	k=k-1
✓7	oi	13	no()!=0	nl()!=0
✓8	ch	7	((	(
✓9	up	4	else(	else[

Thanks to Dale Walker of New York City for the second correction, and again to David McWherter for the other four.

~ 4.2 Functions Omitted from Chapter IV PPS

The version of PPS in Chapter IV does not include the functions `chrdr`, `fofen`, `fread`, `fwrite`, or `fclose`. Appendix C does include these functions. The machine-readable media also include these functions, and may, in fact, include still others as noted in the appropriate installer's guides.

✓4.3 index Documentation

In the documentation for the library function index on page 2-24 of the tiny-c Owner's Manual, the words "OF" and "IN" should be interchanged.

✓4.4 Line 0 Errors

On page 3-2 of the tiny-c Owner's Manual, line 0 of the program buffer in PPS is defined to be just a carriage return. Nevertheless, PPS can produce error messages which reference line 0, especially:

0 -- err 26

<

These error messages actually apply to the console input line and not to the program in the program buffer. Error 26 usually means that you tried to initiate the execution of a function which could not be found in the program buffer. Thanks once more to John Osudar for pointing this out.

tiny-c
Newsletter

DATE: May, 1979
NUMBER: 2

1.0 New Products

The tiny-c interpreter and Program Preparation System are available in six new formats: TRS-80 cassette; CP/M 8" soft-sectored and Micropolis 5" dual or quad density diskette; North Star DOS 5" single density diskette; and a PDP-11 to 8080 cross-assembled version.

The TRS-80 cassette version is recorded in Level II SYSTEM format and includes line printer and graphics support. It also reads and writes EDTASM compatible files. At least 16K bytes of random access memory are recommended for its effective use. The load-and-go TRS-80 cassette of tiny-c costs \$30.

The CP/M and North Star installations are fully interfaced to their respective disk operating systems. The CP/M diskette also contains the source code of the interpreter. The North Star version loads at 2A00. Both the CP/M and North Star diskette versions of tiny-c are \$35.

The PDP-11 to 8080 cross-assembled version of tiny-c consists of the 8080 source code of the tiny-c interpreter on a PDP-11 diskette together with a full set of macros to assemble the 8080 code with the DEC MACRO-11 assembler. The PDP-11/8080 cross-assembled diskette costs \$35.

tiny-c continues to be available for the SOL-20 on Helios diskette and CUTS cassette, on Poly-88 and Tarbell cassettes, and on DEC RT-11 diskette.

2.0 Full C Compiler

A full C compiler for most DEC operating systems is now available through tiny c associates. The compiler, by Whitesmiths, Ltd., handles bit fields and casts and comes with an extensive runtime library. Binary licenses are \$500 and source licenses \$5000. Contact Scott Guthery at (609) 443-3992 for details.

3.0 Computer Shows

tiny c will have booths at the Trenton Computer Festival, the West Coast Computer Faire, and the Philadelphia Computer Show this year. Be sure to stop by and say "Hi".

4.0 Program Preparation System Fixes

Here are some bug fixes and improvements for PPS:

4.1 Improved backward scan

The following change to PPS makes the -n command work with blinding speed. On page 9 of Appendix C, replace the up function with:

```
up [  
  int v(1)  
  if(ln(1)==0) v(0)=1  
  else num(ln+1,v)  
  if((v(0)=10-v(0))<0) v(0)=0  
  so(v)  
]
```

4.2 Problems with too many DELs

There is a bug in the function ss on page 1 of Appendix C which causes problems if too many DELs are typed. The fifth line needs a semicolon so that it reads:

```
else if(b(1)==127)[if(l>0)l=l-1;]
```

This fix is required on all media labeled 80-01-01. Note that the constant 127 is installation dependent so if you find a different constant in your PPS, don't change it.

4.3 Typo in in function

In the in function in Appendix C on page 8, line 31, `ll==ll+1` should read

```
ll=ll+1
```

The machine readable media are correct. Thanks to Robert Pasky of Waltham, Massachusetts, for this one.

4.4 Error return without a close

In the readfile function in Appendix C on page 4, line 9, the "if(k==-1) return t" statement should read

```
if(k==-1){MC(u,6);return t;}
```

so that the unit on which the readfile error was raised is closed before returning. This fix is required on some, but not all, media labeled 80-01-01, and some may already have it. Even though some installations don't need to worry about this it's safer to have it even if it's not required. Thanks to Ray Duncan of La Verne, California, for calling this one to our attention.

4.5 Further Problems with fc

In the fc function in Appendix C on page 6, line 14, the while statement should read:

```
while((r(k=k-1)!=13)if(k<=0)break
```

which stops an occasional PPS abort with a subscript out of range error.

4.6 Null locate

If the PPS command line

```
.1/<CR>
```

is entered, a subscript underflow error is raised due to the check for the caret character at the end of the string to be located. This can be prevented by placing the statement

```
if(ln(3)==0)return
```

as the first executed statement in the function oi in Appendix C on page 7.

4.7 PPS as an application program

A few tiny-c users called in with this problem:

"I was using PPS to edit PPS. The I started the edited PPS to test it. So far, so good. I used this PPS (running as an application) to prepare a small program. Still OK. I even ran the program (as a second level application), and stopped it, edited it, and ran it again. Everything was just fine. Then I aborted the application level PPS to return to the system level one. And the system crashed!!!"

The problem is the endlibrary statement, which is permitted only at the system level. If you want to TEST a PPS at the application level, remove its endlibrary statement. Then test it. Then put the endlibrary statement back before recording it.

Some other notes. To run a PPS at application level, be sure to reduce the pr dimension and, 4 lines later, the lp assignment value. Two or three hundred bytes is enough space in pr to test a new PPS. Remember you have two copies of PPS in RAM and two complete sets of its variables.

If you change the length of the input line ln in its dimensions statement be sure to put the new length in the while condition in the st function.

By the way, before using PPS to edit PPS make sure the original dimension of the pr array is large enough to accomodate PPS itself. PPS is about 5015 bytes so an original dimension of 5200 or more should suffice. Some versions of PPS are delivered with a dimension of 5000 and these will have to be changed using the standard system edit. Remember to change both the dimension of pr and the assignment four lines later.

4.8 Null filename reads and writes

There is currently no check in the readfile or writefile routines for a null (zero length) filename. Some operating systems tolerate this condition, some do not. If you have any problems with a .r or .w command with no filename, you should put a check for a null filename in readfile and writefile.

4.9 PPS extensions

Don de Courcelle of Avenel, New Jersey, sends along the following helpful extensions for PPS. Many thanks to Don and kudos to his emphasis on user oriented computing.

FOR THOSE USERS THAT ARE TIRED OF REFERING BACK TO THE ERROR MESSAGE TABLE IN THE BACK OF CHAPTER 3, I RECOMMEND THESE 2 ROUTINES REPLACE THOSE FOUND IN THE STANDARD PPS (ON PAGES 9 AND 10 OF APPENDIX C). THEY ALSO IMPLIMENT ERROR HANDLING FOR IMMEDIATE LINE-INPUT EXECUTION (FORMERLY REPORTED AS LINE 0 ERRORS WITH NO DISPLAYED TEXT)...

```

WH
[INT F, L, U, B, X
PL "#EXEC FAULT @ LINE"
PN LO; PS" -- ERR: ";
X=ER(0)
IF(X==1) PS"BAD STMT"
ELSE IF(X==2) PS"CURSOR EOF"
ELSE IF(X==3) PS"ID REQD"
ELSE IF(X==5) PS"MISSING ' '"
ELSE IF(X==6) PS"INV INDEX"
ELSE IF(X==7) PS"BAD PTR EXPR"
ELSE IF(X==9) PS"AEXP REQD"
ELSE IF(X==14) PS"BAD ASSIGN"
ELSE IF(X==16) PS"STK OVFL0"
ELSE IF(X==17) PS"FUNC OVFL0"
ELSE IF(X==18) PS"VARI OVFL0"
ELSE IF(X==19) PS"OUT OF MEMORY"
ELSE IF(X==20) PS"STARTUP ERR"
ELSE IF(X==21) PS"WRONG NUM OF PARMS"
ELSE IF(X==22) PS"FUNC BODY ERR"
ELSE IF(X==24) PS"BAD MC"
ELSE PS"UNDEF SYMBOL"
PL""
IF (CU < 0)
[INT I
WHILE( (LL=LL-1) > 0) PUTCHAR( LN(I=I+1) ); PL""
B=64+CU
CU=0
]
ELSE
[CU=CU
F=FC
B=U-F
L=LC
F=F-1
WHILE( (F=F+1) < L) PUTCHAR( PR(F) ); PL""
]
WHILE( (B=B-1) >= 0) PUTCHAR( ' ' )
PS "<--- ERR ---<<"; PL""
]

```

```

ST
[INT LLX, SVCU
LLX=LL
WHILE( LLX <= 64 )
[LN(LLX)=' '
LLX=LLX+1
]
MC ER, LN+1, PR+PE, PR, 11
IF(CU < 0)
[LN(LL)=0 /* RESTORE END OF LINE NULL
SVCU=CU
CU=0
]
ELSE
IF(CU > PE) CU=PE

```



```

LO=COUNTCH(PR, PR+CU-1, 13)
IF( ER(0) )
  IF( ER(0)==99 ) [ PL"#ESC ABORTED"; PL"" ]
  ELSE
    [IF(SVCU != 0) CU=SVCU
    WH
    ]
]

```

FOR THOSE USERS WHO HAVE A CRT TERMINAL, REPLACING THIS ROUTINE FOR THE ONE IN PPS CAN PROVE ENJOYABLE (REFER TO PAGE 1 - APPENDIX C). IT TURNS YOUR "DEL" KEY INTO A GENUINE BACKSPACE !!

```

GS CHAR B(0)
[INT I; CHAR C
I=-1
WHILE (1)
  CC = B(I+1) = MC 2
  IF(C==13) [B(I)=0; RETURN I]
  ELSE
    IF(C==27) [PS" <<DEL<<"; PL""; I=-1]
    ELSE
      IF(C==127)
        IF(I >= 1)
          [PUTCHAR(8)
          PUTCHAR(' ')
          PUTCHAR(8)
          I=I-2
          ]
        ELSE I=I-1
      ]
]

```

FOR THOSE USERS THAT WOULD LIKE TO EDIT LARGER LINES OR FILES USED THESE CHANGES IN YOU PPS (REFER APPENDIX C):

```

>>> ON PAGE 5, DELETE LINE 3>> CHAR FT(20), TT(20)
>>> CHANGE LINE 4 TO: INT FL, TL, LNSZ, PRSZ
>>> CHANGE LINE 5 TO: CHAR LN(LNSZ=80), PR(PRSZ=8000)
>>> INSERT AFTER LINE 5: CHAR FT(LNSZ), TT(LNSZ)
>>> CHANGE LINE 9 TO: LP=PRSZ
>>> IF YOU USE THE NEW "WH" B=LNSZ+CU
ROUTINE GIVEN ABV USE THIS
THIS INSTEAD OF "B=64+CU".
>>> IF YOU USE THE NEW "ST" WHILE( LLX <= LNSZ )
ROUTINE GIVEN ABV USE THIS
TO REPLACE "WHILE(LLX<=64)"
OTHERWISE CHANGE "64" TO
"LNSZ" IN LINE 2 OF PAGE 10.

```

FOR THOSE USERS DESIRING A MORE HUMANIZED ./ COMMAND SUBSTITUTE THIS ROUTINE INTO PPS (APPENDIX C, PAGE 9):

```

FA
[PS"CURRENT LINE:"; PN LO
PS" LINES:"; PN LA
PS" BYTES USED:"; PN PE
PS" BYTES LEFT:"; PN LP-PE; PL""
]

```

THIS CONVENIENCE CHANGE MAKES A SINGLE CARRIAGE-RETURN INPUT AT THE BEGINING OF A LINE ACT EXACTLY LIKE A "+ CARRIAGE-RETURN", INCREMENTING THE LINE POINTER AND DISPLAYING THE NEXT LINE. (REFER TO APPENDIX C).

```

>>> CHANGE LINE 13, PG 5: LL=GS(LN)
>>> AFTER THIS LINE >> ELSE IF(C=='+' ) DO
INSERT THIS LINE: ELSE IF(C==0) NEXT
>>> AFTER FUNCTION "DO" NEXT [ LN(1)=0; DO ]
ON PG 9 INSERT THIS
NEW FUNCTION.

```


5.0 8080 Fixes

Here is a collection of fixes and improvements for the 8080 version of tiny-c.

5.1 Ordering

The POP D and JC PERR are in the wrong order at 21DE. Put the POP D before the JC PERR. This fix is required on all media labeled 80-01-01.

5.2 Typos

The last two listing lines in Appendix A should read

```
30F4      3A 93 20
30F7      C3 59 2D
```

The distributed media are correct. Note that there are three typos corrected here.

Another typo is noted in 5.4a below.

5.3 Misleading comment

In line 217C the comment incorrectly describes the action of DCMF, which does not set Z as described. Delete "Z," from the comment.

5.4 Assembly from Appendix A

For those entering the 8080 source code from Appendix A and assembling it, the following changes may be required depending on the properties of your assembler:

a) At 2782, B7 ORA A,E should be B3 ORA A. Some media require correcting the object code byte.

b) At 2209, MVI 0,B should be MVI B,0

c) The labels D2, D3, D4, DONE, and WHERE are duplicated. Further, PUSH, POP, and OUT are used as labels. Make the following changes:

- in DREM (2131 to 216E) change all occurrences of D2, D3, and D4 to DR2, DR3, and DR4 respectively

- in BLANKS (2243 to 2253) change all occurrences of OUT to BOUT

- in FACTOR (26FB to 282A) change all occurrences of WHERE to FWHERE

-at 2CD8 and 2D00 change DONE to DONEMSG

-change the subroutine names PUSH (21CD) and POP (21A9) to PUSHST and POPST respectively, and reflect these changes in all CALLs to these routines.

Thanks to James A. Petroski for pointing these items out.

5.5 Using the <CR><LF> pair as end-of-line

The following patches make the 8080 interpreter allow either <CR> or the pair <CR><LF> as the end-of-line indicator:

Change REM as follows:

replace	2349	JNZ	REM
	234C	LXI	D,CMNT

with	JZ	RE2
RE3	MOV A,M	
	CPI	0AH ; <LF>
	JNZ	REM
	INX	H
	SHLD	CURSOR
	JMP	REM
RE2	LXI	D,CMNT

and replace	235B	JMP	REM
-------------	------	-----	-----

with	JMP	RE3
------	-----	-----

Change SKIPST as follows:

replace	286D	JZ	SS5
---------	------	----	-----

with	JZ	SS8
------	----	-----

and replace	2883	SHLD	CURSOR
-------------	------	------	--------

with	SS8	SHLD	CURSOR
------	-----	------	--------

5.6 Stack overflows

In the 8080 version if any of the areas STACK, FUN, or VAR is exceeded, a few bytes beyond its Ending address are overwritten. (See Section 6.5.2) This can cause unexpected DONE messages. A small guard area at the end of each space will protect against this.

After calculating the eight allocation addresses, add 5 to ESTACK, 6 to EFUN, and E(hex) to EVAR. So the example on page 6-19 of the tiny-c Owner's Manual would read:

	begin	-end-1+guard
	=====	=====
STACK	1200	ED85
FUN	1280	ED06
VAR	1300	E60E
PR	1A00	C000

There is no way to guard against an overflowing of the 8080 machine stack. Be sure to allocate plenty of RAM for it. 1000 bytes should usually be enough. A highly recursive application may require more however.

6.0 CP/M Fixes

The following fixes should be applied to CP/M disks labeled 80-01-01. Disks labeled 80-01-01b have had these fixes already applied to them.

6.1 Incorrect default DMA address

In CTC.ASM, line 110 should read

```
LXI    D,80H
```

Patch TC.COM as follows:

```
A> ddt tc.com
- 1157
  (erroneous TC.COM will show LXI D,0500)
- a157
157 LXI D,80
15A .
- 1157
  (corrected TC.COM will show LXI D,0080)
- <ctrl>C
A> save 23 tc.com
```

The fifth line from the bottom of the CP/M version of PPS reads

```
k=81+32*(k%4)
```

Change the 81 to a 129.

6.2 Missing right parenthesis

The fourteenth line needs a right parenthesis after the 127. If it is not there, this was the cause of the <ctrl>U blowups.

7.0 Programming Technique

The MC 11 library machine call can be used to overlay large application programs. The general scheme is as follows. Prepare a "root segment" file which looks like this:

```
library routines used by most overlays
endlibrary statement
super globals ... communication between overlays
main program that does overlay work
```

One of the super globals is a filename. The main program works as follows:

```
set filename to 1st overlay
while(filename is not finish signal) [
    read the named file
    use MC 11 to execute it
]
```

Overlays are written with the following protocol: to cause control to pass to another overlay, put that overlay's filename into the filename super global and return. To cause the whole program to stop, put the finish signal in the filename super global and return.

Data can be passed among the overlays using other supersglobals.

An overlay cannot have an endlibrary statement in it (see 4.7 above).

If somebody comes up with a specific implementation or application of overlays we will pass it along in the next Newsletter.



tiny c associates
post office box 269
holmdel, new jersey 07733
201-671-2296

tiny-c newsletter #3

To those with Owner's Manuals whose numbers are larger than 972 this will be your first newsletter. The modifications noted in the first two newsletters were included in the second printing.

Besides these additional modifications that have been brought to our attention, we are using this opportunity to announce our new tiny-c TWO compiler. The 20% discount is good for an unlimited period. Tiny-c TWO is currently available only on 8" CP/M diskette, but other versions should be ready soon. Watch our ads in Dr. Dobb's and Creative Computing for additional versions.

PPS use of additional memory

Modifications to PPS to use additional memory were not too well explained in Chapter 6. Two lines must be changed. In the crunched PPS about five lines after the "endlibrary" statement is:

```
char ln (120), pr(DDDD)
```

About four lines later is:

```
lp=DDDD
```

These two occurrences of the decimal number DDDD must be edited to inform PPS that it has more room to work in. This is in addition to changes made to BPR and EPR as explained in section 6.5.2.

A reasonable dimension for PR in PPS is:

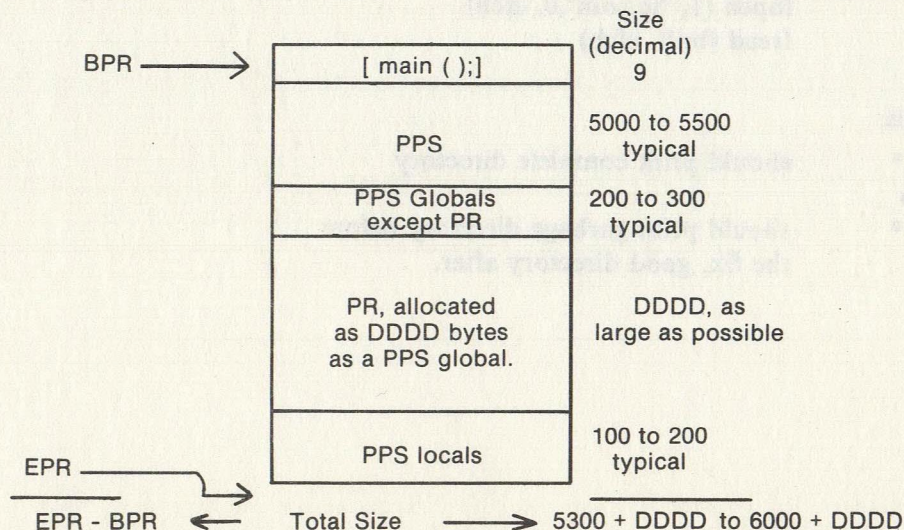
```
PRSPACE - PPSLENGTH - 500 (dec)
```

PPS lengths vary depending on installation. Most are around 5000 to 5500 bytes. So try:

```
PRSPACE - 6000(dec)
```

PRSPACE is the number of bytes allocated between BPR and EPR, expressed in decimal.

Example: On page 6-19 (section 6.5.2) PR in the interpreter runs from 1A00 to 4000. This is 2600 hex or 9728 decimal. DDDD can be 9728 - 6000, or 3700 in round numbers. If your PPS is smaller than 5500, DDDD can be made correspondingly larger. The following picture should help illustrate this:



8080 SKIPST problem:

In Appendix A at 286A, the code starting at label SS4 should read:

```
SS4  MOV A,M
      CPI ASCRET

SS4  MOV A,M
      CPI ASCRET
      JZ SS8
      CPI ']' ;new line
      JZ SS8 ;new line
      CPI ';'
      JZ SS5

SS5  INX H
SS8  SHLD CURSOR ;new label
      JMP REM
```

As two new lines are being added, either a patch technique or reassembly is required. Not that the first JZ SS8 and the label SS8 were added in a fix in Newsletter #2, paragraph 5.5.

Restoring CP/M DMA area:

If an application program fails to close all files the CP/M DMA area may not be restored to 80H. This shows up when the .dir command is given from PPS, and the directory printout is crazy. The names of four or fewer files are printed out over and over again. The fix is simple. The installation vector at ORG+29H has.

```
PRDONE  NOP
         NOP
         RET
```

Change this to:

```
PRDONE JUMP CPMBUFF
```

and re-assemble.

This program can be used as a test:

```
bomb
[
    char efc (50), buff (128)
    fopen (1, "tc.com", 0, efc)
    fread (buff, efc)
]
```

In PPS do this:

.dir *.*	should print complete directory
.bomb	
.dir *.*	should print garbage directory before the fix, good directory after.